

AD-A269 262



1
H8

A Report
Prepared For

CONTEL CORPORATION

DTIC
ELECTE
SEP 10 1993
S A D

RECOMMENDATIONS FOR AN INITIAL SET OF SOFTWARE METRICS

CTC-TR-89-017

December 12, 1989

93-21027

This document has been approved
for public release and sale; its
distribution is unlimited.

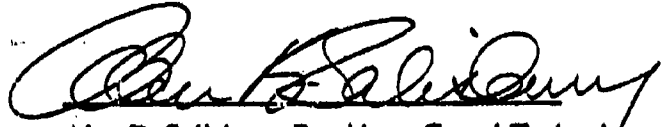
Prepared by:
Shari Lawrence Pfleeger

Contel Technology Center
15000 Conference Center Drive
P.O. Box 10814
Chantilly, Virginia 22021-3808

9/14/93

()
()
()
()
()
()
()
()
()
()

Released for publication



Alan B. Salisbury, President, Contel Technology Center

© 1989, Contel Corporation. All rights reserved.

ABSTRACT

The Contel Technology Center's Software Engineering Laboratory (SEL) has as one of its goals the improvement of productivity and quality throughout Contel Corporation. The SEL's Process and Metrics Project addresses that goal in part by recommending metrics to be collected on each software development project throughout the corporation. This report suggests an initial set of metrics for which data are to be collected and analyzed, based on a process maturity framework developed at the Software Engineering Institute. Metrics are to be implemented step by step in four phases, corresponding to the maturity of the development process. Phase 1 metrics focus on project management. Phase 2 metrics measure the products produced during development, while Phase 3 metrics capture characteristics of the development process itself. Phase 4 adds feedback loops to the process metrics, so that metrics play an active and ongoing role in assessing and controlling the development process from the very beginning of a project.

The classes of metrics for each phase are listed and discussed in terms of what to collect, how to collect it, and the expected behavior and benefits of each. After defining a set of metrics for each phase, the report explains how the initial metrics set is to be implemented in a Contel business unit. The explanation includes a discussion of the role of the Process and Metrics Project, including the products and services available to every Contel organization.

Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By <i>per lte</i>	
Distribution	
Availability Codes	
Dist	Avail. or Special
A-1	

DO NOT WRITE IN THESE SPACES

EXECUTIVE SUMMARY

Dozens, if not hundreds, of software metrics are described in the software engineering literature. The metrics chosen for a particular project play a major role in the degree to which the project can be controlled, but deciding which metrics to use is difficult. We can evaluate the purpose and utility of each metric only in light of the needs and desires of the development organization. Thus, we should collect data and analyze software metrics in the broad context of the software development process and with an eye toward understanding and improvement. It is for this reason that we have chosen a process maturity framework in which to place software metrics. Originating at the Software Engineering Institute in Pittsburgh, Pennsylvania, process maturity describes a set of levels at which the development process takes place. Only when the development process possesses a particular structure or organization does it make sense to collect certain kinds of metrics.

Thus, rather than recommend a large (and probably unwieldy) set of metrics to be collected on every project throughout an organization, we recommend that metrics be divided into four phases, where each phase is based on the amount of information made available by the development process. Metrics collection begins at Phase 1, moving on to the other phases only when dictated by a process that can support it.

PROCESS MATURITY LEVELS

The concept of process maturity is based on the notion that some development processes provide more structure or control than others. This notion does not judge the quality of the particular process. Instead, it provides a framework in which to depict the several types of processes and to evaluate what kinds of metrics are best suited for collection in each type.

Figure 0.1 depicts the five levels of process and their characteristics.

93-17193



Level	Characteristics	Measurement
5. Optimizing	Improvement fed back to process	Process + feedback for changing process
4. Managed	Measured process (quantitative)	Process + feedback for control
3. Defined	Process defined, institutionalized	Product
2. Repeatable	Process dependent on individuals	Project
1. Initial	Ad hoc/chaotic	Preliminary project (baseline)

Figure 0.1 – Process Maturity Levels Related to Metrics

Level 1: Initial Process

The first level of process is termed *initial* and is characterized by an ad hoc approach to the software development process. That is, the inputs to the process are defined, and the outputs are expected, but the transition from input to output is undefined and uncontrolled. For this level of process, the collection of metrics is difficult. Only baseline project metrics should be gathered, to form a basis for comparison as maturity increases. Rather than concentrate on metrics and their meanings, the developers should focus on imposing more structure and control on the process.

Level 2: Repeatable Process

The second process level, called *repeatable*, identifies input, output and some controlling mechanism. The control is usually in the form of project management, where the cost and schedule can be tracked as the project progresses. Figure 0.2 depicts a repeatable process as an SADT diagram, where the incoming arrow on the left shows the input, the outgoing arrow on the right the output, and the arrow from the top the control. Only project-related

metrics make sense at this level, since the activities within the actual transition from input to output are not available to be measured. Thus, for a repeatable process, we can measure factors such as the amount of effort needed to develop a system, the duration of the project, the volatility of the requirements, and the overall project cost. The output can be measured in terms of its functional size.

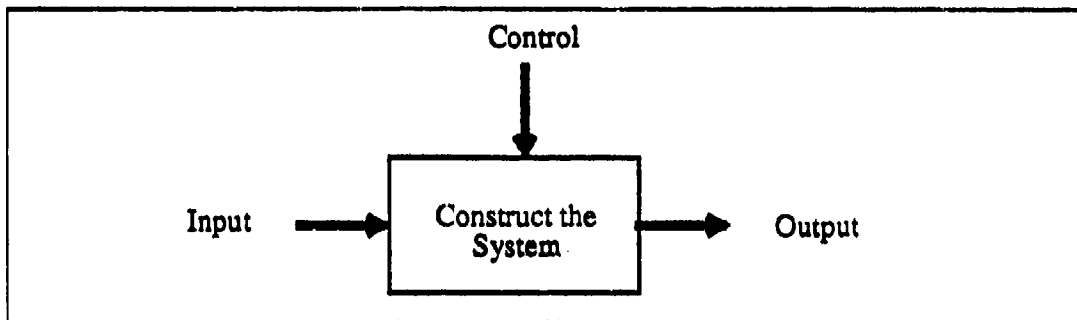


Figure 0.2 – SADT Diagram of Repeatable Process

We recommend the following types of measures at this level:

- Software size
- Personnel effort
- Requirements volatility

Several additional metrics may be desirable, depending on the characteristics of the project and the needs of project management. Many studies of project cost indicate that experience and employee turnover can have a significant impact on overall project cost. Thus, the following items can be added to the Phase 1 metrics set at the discretion of management.

- Experience
- Employee turnover

Level 3: Defined Process

The third process level is called *defined*, because the activities of the process are clearly defined, as depicted in Figure 0.3.

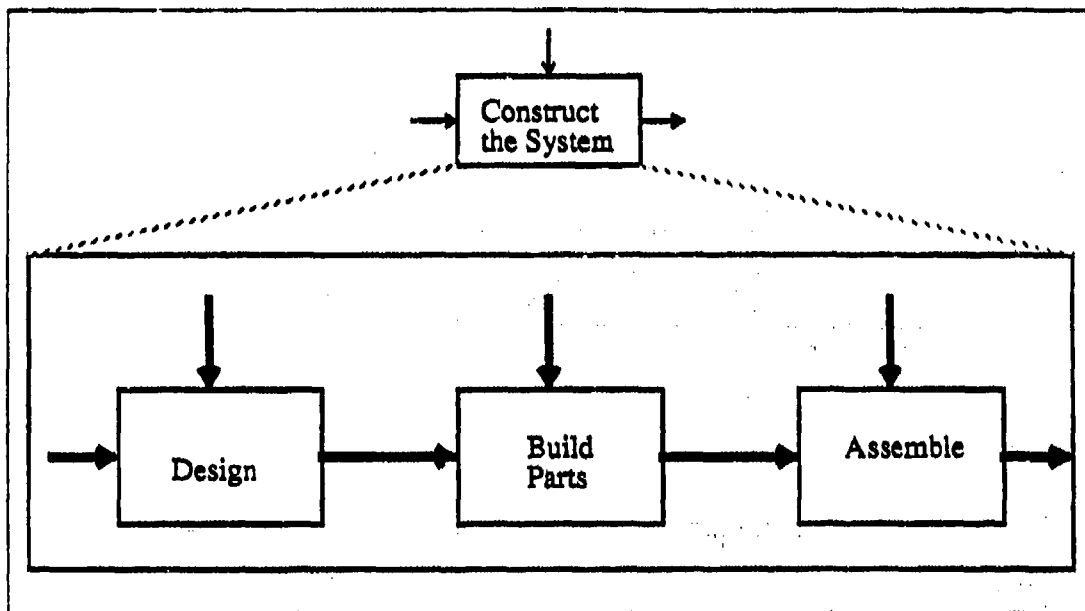


Figure 0.3 – SADT Diagram of Defined Process

This additional structure means that we can examine the input to and output from each functional activity performed during development. The box of Figure 0.2 can be exploded to view the activities necessary to construct the final system. Figure 0.3 describes three typical activities: design, build parts and assemble. However, different processes may be partitioned into more or fewer distinct functions or activities. Figure 0.4 is an example process, displaying the details of input, output and control for each activity.

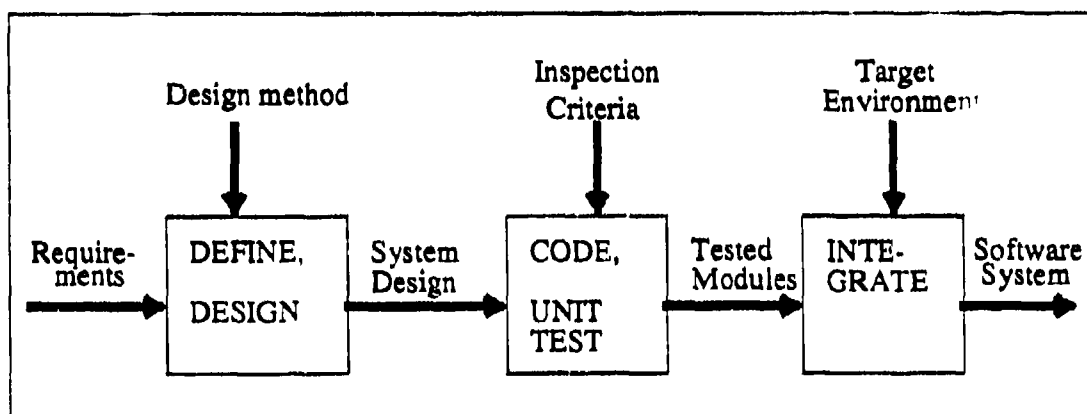


Figure 0.4 – Example of a Particular Defined Process

Because the activities are delineated and distinguished from one another, the products from each activity can be measured and assessed. In particular, project managers can look at the complexity of each product. That is, we can examine the complexity of the requirements, design, code and test plans, and assess the quality of the requirements, design, code and testing.

In terms of complexity, we suggest that the following items be examined first for a defined process:

- Requirements complexity
- Design complexity
- Code complexity
- Test complexity

One perspective from which to view the quality of the products is examination of the number of faults in each product and the density of defects overall. In addition, the thoroughness of testing can be assessed. Thus, our recommended quality metrics include:

- Defects discovered
- Defects discovered per unit size (defect density)
- Requirements faults discovered
- Design faults discovered
- Code faults discovered

We emphasize that this set does not represent the full spectrum of quality measures that can be employed. Issues of maintainability, utility, ease of use, and other aspects of quality software are not addressed by defect counts. However, defect analysis is relatively easy to implement, and it provides a wide spectrum of useful information about the reliability of the software and the thoroughness of testing.

An additional product metric may be desirable. When customer requirements dictate that significant amounts of documentation be written (as often happens on government contracts), the number of pages of documentation may be a desirable characteristic to track and correlate with effort or duration. Thus, the set of product metrics may also include

- Pages of documentation

Level 4: Managed Process

The *managed* process is the fourth level of maturity; enough control is available for each process activity to allow us to measure overall process characteristics. As shown in Figure 0.5, this level allows measurements to be made across activities. Because activities can be compared and contrasted, the effects of changes in one activity can be tracked in the others. For example, we can measure and evaluate the effects of major process factors such as reuse, defect-driven testing, and configuration management. The measures collected are used to control and stabilize the process, so that productivity and quality match expectations.

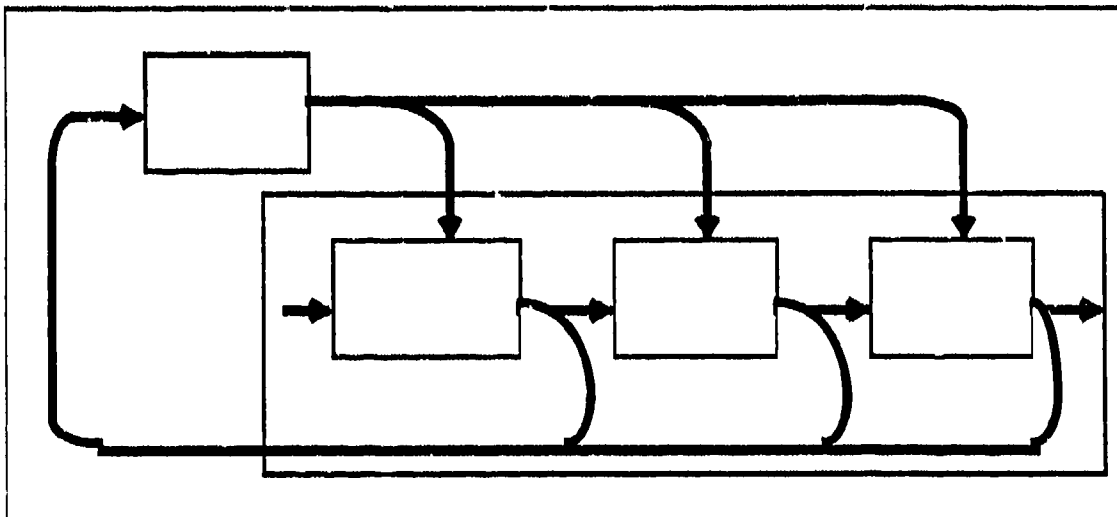


Figure 0.5 – SADT Diagram of Managed Process

Figure 0.6 illustrates how a particular managed process might look. Metrics are used in feedback loops to report on the number of design defects and on the number and types of problems encountered with specific versions of the system. Then, project management uses the metrics information to make decisions about course corrections.

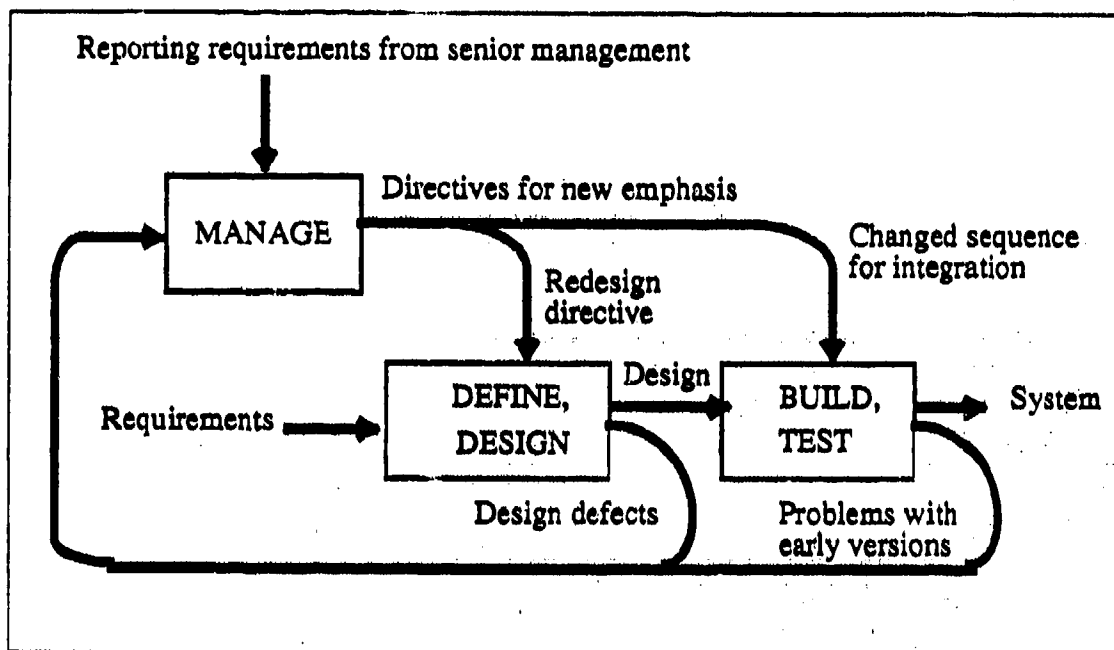


Figure 0.6 – Example of a Managed Process

If the maturity of the process has reached the managed level, then process-wide metrics can be collected and analyzed. These metrics reflect characteristics of the overall process and of the interaction among components of the process. A distinguishing characteristic of a managed process is that the development of software can be carefully controlled. Thus, a major characteristic of the recommended metrics is that they help management to control the development process.

We recommend that the following types of data be collected for a managed process. In some cases, the actual metrics must be defined and analyzed to suit the development organization.

- Process type
- Amount of producer reuse
- Amount of consumer reuse
- Defect identification
- Use of defect density model for testing

- Use of configuration management
- Module completion over time
- Capital intensity

All of the process metrics described above are to be used in concert with the metrics discussed in earlier sections. Relationships can be determined between product characteristics and process variables to assess whether certain processes or aspects of the process are effective at meeting productivity or quality goals. The list of process measures is by no means complete. It is suggested only as an initial attempt to capture important information about the process itself.

Level 5: Optimizing Process

An *optimizing* process is the ultimate level of process maturity; it is depicted in Figure 0.7. Here, measurements are fed back to project management as development progresses, so that decisions about both activities and the process itself can be made based in part on the metrics.

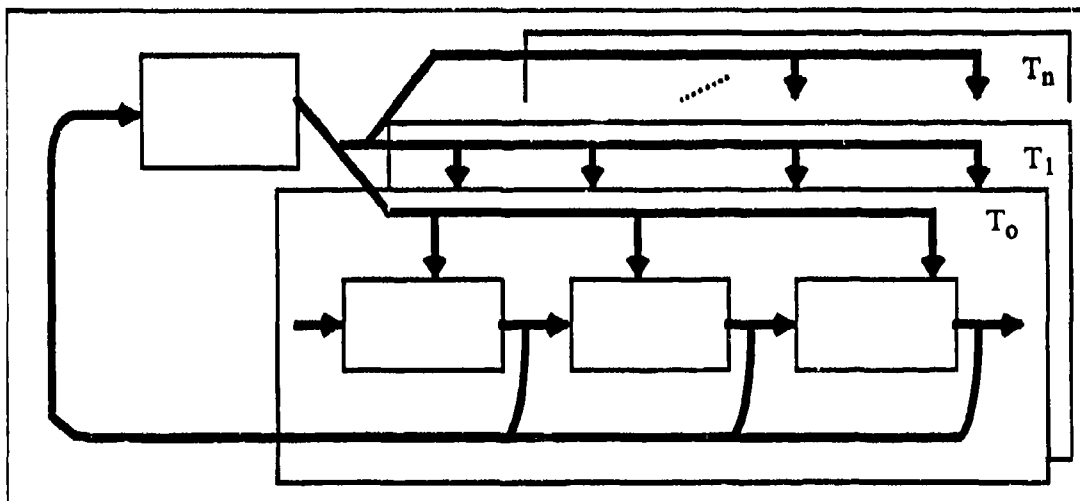


Figure 0.7 – SADT Diagram of Optimizing Process

This dynamic tailoring of the process to the situation is indicated in the figure by the collection of process boxes labelled T_0 , T_1 , ..., T_n . At time T_0 , the process is as represented by box T_0 . However, at time T_1 , management has the option of revising or changing the

overall process. For example, the project manager may begin development with a standard waterfall approach. As requirements are defined and design is begun, metrics may indicate a high degree of uncertainty in the requirements. Based on this information, the process may change to one that prototypes the requirements and the design, so that we can resolve some of the uncertainty before substantial investment is made in implementation of the current design. In this way, an optimizing process gives maximum flexibility to the development. Metrics act as sensors and monitors, and the process is not only under control but is dynamic, too.

Studies by the Software Engineering Institute of 113 software development organizations report that 85% of those surveyed were at level 1, 14% were at level 2, and 1% were at level 3. That is, none of the development projects had reached levels 4 or 5: the managed or optimizing levels. Based on these results, and the highly unlikely prospect of finding a level 5 project, our recommendations for initial metrics include only the first four levels.

STEPS TO TAKE IN USING METRICS

Metrics are most useful only when implemented in a careful sequence of process and metrics activities. These activities lay the groundwork for effective project management by evaluating the needs and characteristics of development before the appropriate phase of metrics collection is identified. The typical development organization should take the following steps:

- **Assess the process:** Working with a set of guidelines or with the Process and Metrics team, determine the level of process maturity achievable (for a proposed project) or implemented (for an on-going one).
- **Determine the appropriate phase of metrics collection:** Once the process level is known, decide which phase of metrics collection is most appropriate. This decision may require further consultation with the Process and Metrics team.
- **Recommend metrics, tools, techniques:** When the types of metrics are determined, identify tools and techniques to be used on the project. Choose these tools and techniques with the overall goals of the project in mind. Whenever possible, implement automated support for metrics collection and analysis as part of the project development environment. It is essential that metrics collection and analysis

not impede the primary development activities; thus, metrics collection and analysis should be as unobtrusive as possible.

- **Estimate project cost and schedule:** Having determined the process level and selected the development environment, estimate the cost and schedule of the project. By using measures implemented at Phase 1, continue to monitor the actual cost and schedule during development.
- **Collect appropriate level of metrics:** Oversee the collection of metrics.
- **Construct project database:** Design, develop and populate a project database of metrics data. This database can be used for analysis, where appropriate. Track the value of metrics over time to understand how project and product characteristics change.
- **Evaluate cost and schedule:** When the project is complete, evaluate the initial estimates of cost and schedule for accuracy. Determine which of the factors may account for discrepancies between predicted and actual values.
- **Evaluate productivity and quality:** Make an overall assessment of project productivity and product quality based on the metrics available.
- **Form a basis for future estimates:** Incorporate the project metrics in a corporate or organizational metrics database. This larger database can provide historical information as a basis for estimation on future projects. In addition, the database can be used to suggest the most appropriate tools and techniques for proposed projects.

EXPECTED BENEFITS

The increased understanding of the process and control of the project offered by a process maturity approach outweigh the effort needed to capture, store and analyze the information required. Objective evaluation of any new technique, tool or method is impossible without quantitative data describing its effect. Thus, the use of metrics in a process maturity framework should result in:

- **Enhanced understanding of the process**

- Increased control of the process
- A clear migration path to more mature process levels
- More accurate estimates of project cost and schedule
- More objective evaluations of changes in technique, tool or method
- More accurate estimates of the effects of changes on project cost and schedule

TABLE OF CONTENTS

<u>Section</u>	<u>Title</u>	<u>Page</u>
	ABSTRACT.....	iii
	EXECUTIVE SUMMARY.....	iv
	TABLE OF CONTENTS	xv
	LIST OF FIGURES	xvii
1	INTRODUCTION	1-1
1.1	Why Metrics?	1-1
1.2	Purpose and Goals of the Software Metrics Program.....	1-1
1.3	Evolution of the Software Metrics Recommendations.....	1-3
1.4	Description of the Remainder of the Document	1-4
2	EMBEDDING SOFTWARE METRICS IN A PROCESS MATURITY FRAMEWORK	2-1
2.1	Rationale	2-1
2.2	Process Maturity Levels	2-1
2.3	Expected Benefits	2-7
3	INITIAL SET OF RECOMMENDATIONS	3-1
3.1	Format of Recommendations	3-1
3.2	Phase 1: Project Metrics	3-1
3.2.1	Type: Software Size.....	3-2
3.2.2	Type: Personnel Effort	3-7
3.2.3	Type: Requirements Volatility	3-10
3.2.4	Possible Additional Project Management Metrics	3-12
3.3	Phase 2: Product Metrics	3-14
3.3.1	Type: Requirements Complexity	3-15
3.3.2	Type: Design Complexity	3-17
3.3.3	Type: Code Complexity	3-21

TABLE OF CONTENTS (Cont.)

<u>Section</u>	<u>Title</u>	<u>Page</u>
3.3.4	Type: Test Complexity	3-24
3.3.5	Type: Defects Discovered	3-29
3.3.6	Type: Requirements Faults Discovered	3-32
3.3.7	Type: Design Faults Discovered	3-33
3.3.8	Type: Code Faults Discovered	3-34
3.3.9	Possible Additional Product Metrics.....	3-34
3.4	Phase 3: Process Metrics	3-34
3.5	Phase 4: Process Metrics with Feedback for Change	3-37
4	USING THE METRICS SETS	4-1
4.1	Steps to Take in Using the Metrics Sets	4-1
4.2	Benefits of Using the Metrics Sets	4-3
A	APPENDIX A -- REFERENCES	A-1

LIST OF FIGURES

<u>Figure</u>	<u>Title</u>	<u>Page</u>
2.1	Levels of Process and Their Characteristics	2-2
2.2	SADT Diagram of Repeatable Process	2-3
2.3	SADT Diagram of Defined Process	2-4
2.4	Example of a Particular Defined Process	2-4
2.5	SADT Diagram of Managed Process	2-5
2.6	Example of a Managed Process	2-6
2.7	SADT Diagram of Optimizing Process	2-7
3.1	Program with Five Lines of Code.....	3-2
3.2	Function Point Computation Chart	3-3
3.3	Complexity Adjustments for Function Points	3-4
3.4	Graph of Size Estimates Versus Actuals	3-6
3.5	Estimates of Planned Person-Months Versus Actual and Reported.....	3-9
3.6	Graph of Project Duration	3-10
3.7	Graph of Requirements and Changes	3-12
3.8	Graph of Experience vs. Productivity	3-13
3.9	Example Requirements with Objects and Actions	3-15
3.10	Graph of Requirements and Complexity	3-17
3.11	Graph with Complexity = 5	3-19
3.12	Graph of Design Complexity Compared with Total Requirements	3-20
3.13	Graph with Complexity = 5	3-22
3.14	Graph of Code Complexity Compared with Total Requirements	3-23

LIST OF FIGURES (Cont.)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
3.15	Graph of Code Complexity Compared with Average Complexity	3-24
3.16	Sample FORTRAN Program	3-25
3.17	Flow Chart of FORTRAN Program	3-26
3.18	Graph of Test Complexity Compared with Requirements Complexity	3-28
3.19	Graph of Defects Discovered Over Time	3-30
3.20	Graph of Percentage of Defects Corrected Over Time	3-31

SECTION 1

INTRODUCTION

1.1 WHY METRICS?

Software plays a major role at Contel, from the systems used to assign new telephone services to the tracking and routing done in support of communications satellites. Corporate success depends on the quality of Contel's products and on Contel's ability to respond to its customers in a timely fashion and at a reasonable cost. Thus, management and control of Contel's software development are paramount in assuring that software products are built on time, within budget and in accordance with a stringent set of quality goals.

Software metrics are essential to understanding, managing and controlling the development process. Quantitative characterization of various aspects of development involves a deep understanding of software development activities and their interrelationships. In turn, the measures that result can be used to set goals for productivity and quality and to establish a baseline against which improvements are compared. Measurements examined during development can point to "hot spots" that need further attention, analysis or testing. Projections and predictions about future projects can be made based on data collected from past projects. Assessments can be made of appropriate tools and strategies, and the development process and environment can be tailored to the situation at hand.

This technical report describes a program for the incorporation of metrics in the software development process at Contel. As part of the Process and Metrics Project, the Software Metrics Program provides both services and products to facilitate the use of metrics throughout the corporation.

1.2 PURPOSE AND GOALS OF THE SOFTWARE METRICS PROGRAM

The Contel Technology Center's Software Engineering Laboratory (SEL) has as one of its goals the improvement of productivity and quality throughout Contel Corporation. The SEL's Process and Metrics Project addresses that goal in part by recommending metrics to be collected on each of Contel's software development projects. These metrics can be help:

- Determine the amount and kind of software development being done at Contel (that is, to find out *what* we are doing)
- Characterize the approaches to software development and the environments in which development is being done (that is, to find out *how* we are doing it)
- Assess productivity and quality for current development projects (that is, to find out *how well* we are doing it)
- Predict and guide the productivity and quality for proposed development projects (that is, to *control* what we do in the future)

Correspondingly, there are two different contexts in which to analyze the metrics: as indicators of status on current projects, and as predictors for future projects. Databases of status information from a collection of existing projects can be amalgamated into a corporate historical database from which predictions can be made for projects being planned. An individual project can measure and track items such as effort expended on the project, number of lines of code produced, number of problems reported, and so on. In addition, input to an estimation tool can be stored, including descriptions of the project team, development environment, constraints imposed by the requirements, and projected size of the final product. These characteristics and the resultant output can be tracked and compared with final measurements to determine not only the accuracy of the estimation tool but also the degree to which the final characteristics match those intended initially.

The corporate historical database can be used in many ways to plan for future development. For example, data can be analyzed to determine the average productivity on a project and how that productivity is affected by the use of different processes, tools and methods. Trade-offs among tools and methods can be determined, so that the best tools and techniques can be chosen for a project based on their success on similar projects in the past. Finally, the cost and schedule of the planned project can be estimated based on cost and schedule information from previous, similar ones.

To support a corporate metrics effort, the Process and Metrics Project team keeps abreast of metrics-related methods, tools and techniques that can assist the business units in their development activities. In particular, the Project acts as a repository of information about

automated metrics data collection and analysis tools, project management software, and cost and schedule estimation tools and techniques.

1.3 EVOLUTION OF THE SOFTWARE METRICS RECOMMENDATIONS

By supplementing the repository of tools and techniques with basic and applied research, the Process and Metrics Project plans to offer a variety of services and products to Contel business units in response to their metrics needs. Over the next three to five years, the products and services will include:

Products:

- A recommended set of metrics to be collected company-wide
- Metrics analysis tools
- Project metrics databases and a corporate historical database
- A decision support tool for assisting project managers in defining, adapting, implementing, measuring and controlling a project's progress

Services:

- Process maturity assessment
- Assistance in developing action plans
- Recommendations for metrics data collection and analysis tools, cost estimation tools and traceability tools
- Seminars, training sessions and technical reports on metrics, process maturity, cost estimation, and software quality and productivity

The initial set of metrics described herein is preliminary to a larger, more comprehensive set of metrics recommendations. In fact, the final set of recommended metrics suggested by the Software Metrics Program will be the result of a multi-stage process. First, the business goals, organizations and commitments of the corporation were reviewed and discussed. Next, the initial metrics set was proposed. This initial set is being collected on

a set of pilot projects spread among several business units. The use of the metrics on these projects will be evaluated in light of the following issues:

- Are the metrics definitions realistic?
- Are the metrics definitions applicable to all projects?
- Are data collection and analysis an unreasonable burden to the projects?
- Are the metrics data useful?

After the results are analyzed, a final recommendation is expected to be made in late 1990.

1.4 DESCRIPTION OF THE REMAINDER OF THE DOCUMENT

The collection of metrics makes sense only when associated with a process context or framework. In other words, data should be collected and analyzed only when they make sense for the software development process being used. This report suggests an initial set of metrics for which data are to be collected and analyzed based on a process maturity framework developed at the Software Engineering Institute¹. The following sections of this document present and explain the process maturity framework, relating the types of metrics to be collected with each level of the hierarchy. As will be shown, the maturity of the process corresponds loosely to the degree to which the process activities are well-defined and controllable. Thus, metrics are to be implemented step by step in four phases, corresponding to the maturity of the development process.

- Phase 1 metrics focus on project management.
- Phase 2 metrics measure the products produced during development.
- Phase 3 metrics capture characteristics of the development process itself, including feedback to control the process.
- Phase 4 adds more feedback loops, so that metrics play an active and ongoing role, not only in assessing and controlling the development process from the very beginning of a project but also in dynamically changing the process itself if need be.

The classes of metrics for each phase are listed and discussed in terms of what to collect, how to collect it, and the expected benefits of each class. After defining a set of metrics for each phase, the report explains how the initial metrics set is to be implemented in a Contel business unit.

EMBEDDING SOFTWARE METRICS IN A PROCESS MATURITY FRAMEWORK

2.1 RATIONALE

Dozens, if not hundreds, of software metrics are described in the software engineering literature. The purpose and utility of each can be evaluated only in light of the needs and desires of the development organization that uses it. Thus, the collection of data and the analysis of software metrics must be performed in the broad context of the software development process and with an eye toward understanding and improvement. It is for this reason that the CTC Software Engineering Laboratory's Process and Metrics Project has chosen a process maturity framework in which to place software metrics. Originating at the Software Engineering Institute, a federally funded research and development center in Pittsburgh, Pennsylvania, process maturity describes a set of levels at which the development process takes place. Only when the development process possesses a particular structure or organization does it make sense to collect certain kinds of metrics.

Therefore, rather than recommend a large (and probably unwieldy) set of metrics to be collected on every project throughout the corporation, the initial set of recommended metrics is divided into four phases. The business unit adopting the recommendations begins at Phase 1, moving on to the other phases only when dictated by a process that can support such metrics collection. The remainder of this section explores the idea of process maturity and explains how process maturity levels are integrated with metrics collection.

2.2 PROCESS MATURITY LEVELS

The concept of process maturity is based on the notion that some development processes provide more structure or control than others. This notion provides a framework in which to depict the several types of processes and evaluate what kinds of metrics are best suited for collection in each type. Figure 2.1 depicts the five levels of process and their characteristics.

Level	Characteristics	Measurement
5. Optimizing	Improvement fed back to process	Process + feedback for changing process
4. Managed	Measured process (quantitative)	Process + feedback for control
3. Defined	Process defined, institutionalized	Product
2. Repeatable	Process dependent on individuals	Project
1. Initial	Ad hoc/chaotic	Preliminary project (baseline)

Figure 2.1 – Levels of Process and Their Characteristics

The first level of process is termed *initial*. It is characterized by an ad hoc approach to the software development process. That is, the inputs to the process are defined, and the outputs are expected, but the transition from input to output is undefined. Similar projects may vary widely in their productivity and quality characteristics, due to lack of adequate structure and control. For this level of process, the collection of metrics is difficult. Developers should focus on imposing more structure on the process. At the same time, however, preliminary project measurements can be taken to form a baseline for later comparison. That is, as the process improves, the degree of improvement can be demonstrated in part by comparing new project measurements with the baseline ones.

The second process level, called *repeatable*, identifies input, output and some controlling mechanism. The control is usually in the form of project management, where the cost and schedule can be tracked as the project progresses. Figure 2.2 depicts a repeatable process as an SADT diagram, where the incoming arrow on the left shows the input, the outgoing arrow on the right the output, and the arrow from the top the control of the process. For example, the requirements may be the input to the process, with the software system as

output. The control arrow represents constraints, such as schedule, budget, tools, standards and other management control directives.

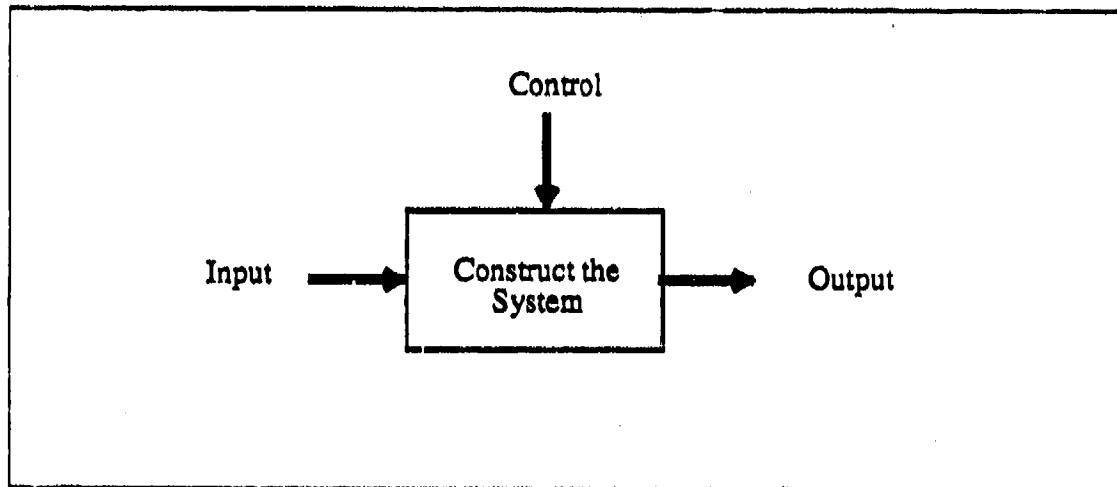


Figure 2.2 – SADT Diagram of Repeatable Process

Project-related metrics make sense at this level, since the activities within the actual transition from input to output are not available to be measured. Thus, measurements can be made of the amount of effort needed to develop a system, the duration of the project, the volatility of the requirements, and the overall project cost. The output can be measured in terms of its physical or functional size, and the resources used to produce that output can be viewed relative to size to yield productivity.

The third process level is called *defined*, because the activities of the process are clearly defined, as depicted in Figure 2.3. This additional structure means that we can examine the input to and output from each functional activity performed during development. The box of Figure 2.2 can be exploded to view the activities necessary to construct the final system. Figure 2.3 describes three typical activities: design, build parts and assemble.

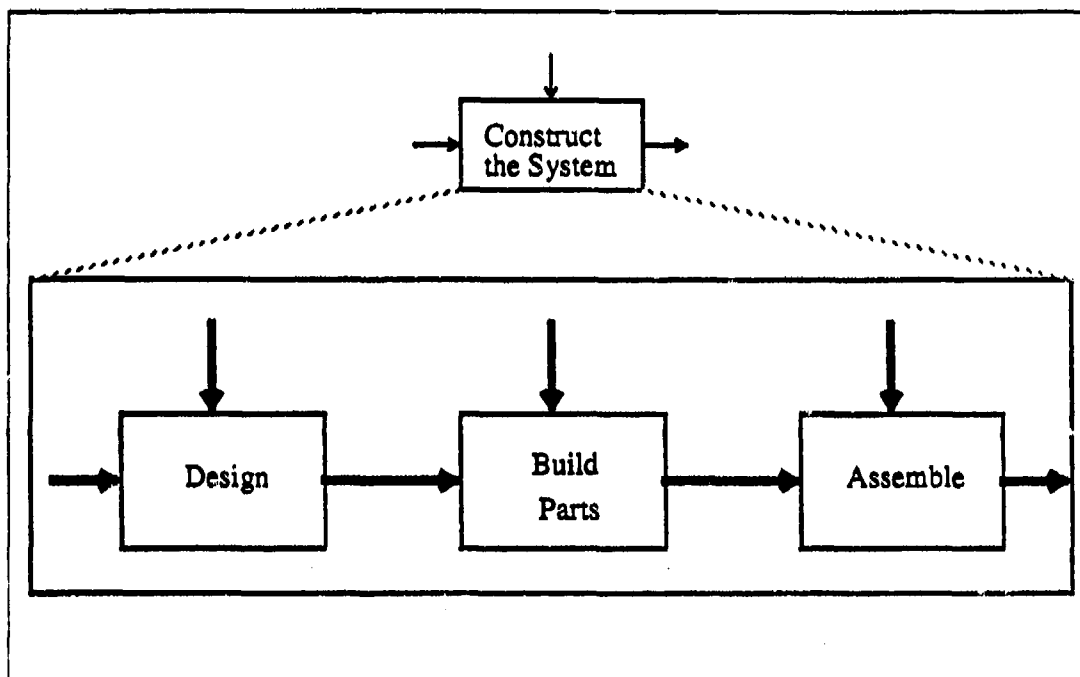


Figure 2.3 – SADT Diagram of Defined Process

However, different processes may be partitioned into more or fewer distinct functions or activities. Figure 2.4 displays an example of a particular process; it shows details of the input, output and control for each activity of the process.

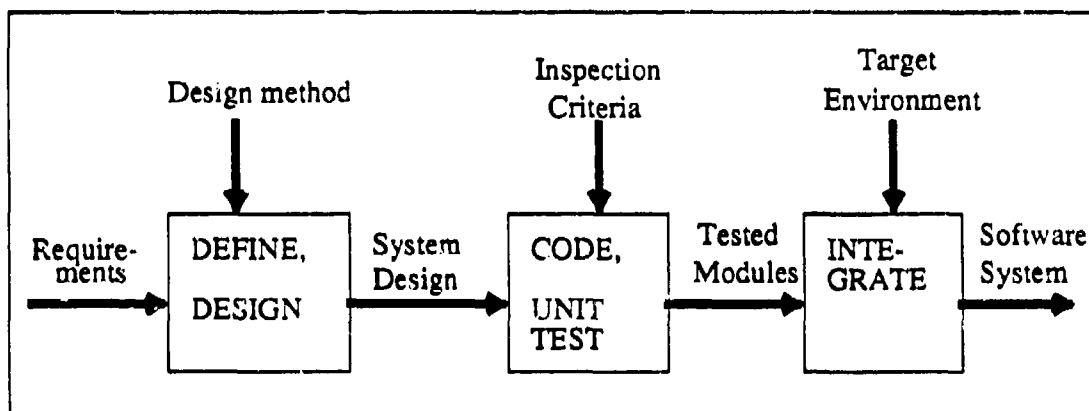


Figure 2.4 – Example of a Particular Defined Process

Because the activities are delineated and distinguished from one another, the products from each activity can be measured and assessed. In particular, project managers can look at the

complexity of each product. That is, the complexity of the requirements, design, code and test plans can be examined, and the quality of the requirements, design, code and testing can be assessed.

The *managed* process is the fourth level of maturity, where enough control is available for each process activity to allow overall process characteristics to be measured. As shown in Figure 2.5, this level allows measurements to be made across activities. Because activities can be compared and contrasted, the effects of changes in one activity can be tracked in the others. For example, the effects of major process factors such as reuse, defect-driven testing, and configuration management can be measured and evaluated. The measures collected are used to control and stabilize the process, so that productivity and quality match expectations.

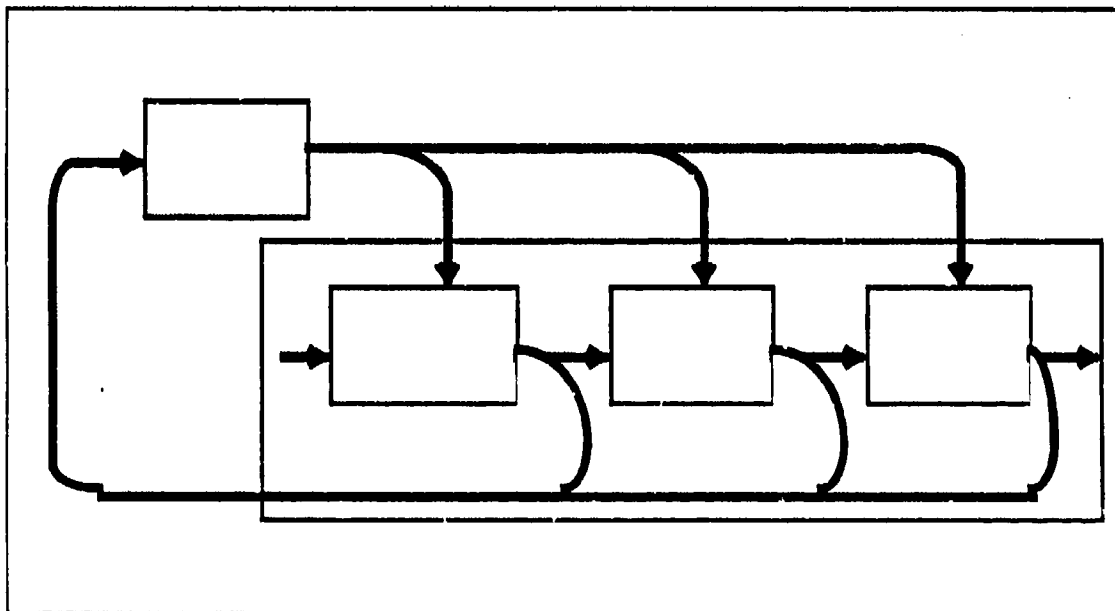


Figure 2.5 – SADT Diagram of Managed Process

Figure 2.6 illustrates how a particular managed process might look. Metrics are used in feedback loops to report on the number of design defects and on the number and types of problems encountered with specific versions of the system. Then, project management uses the metrics information to make decisions about "course corrections".

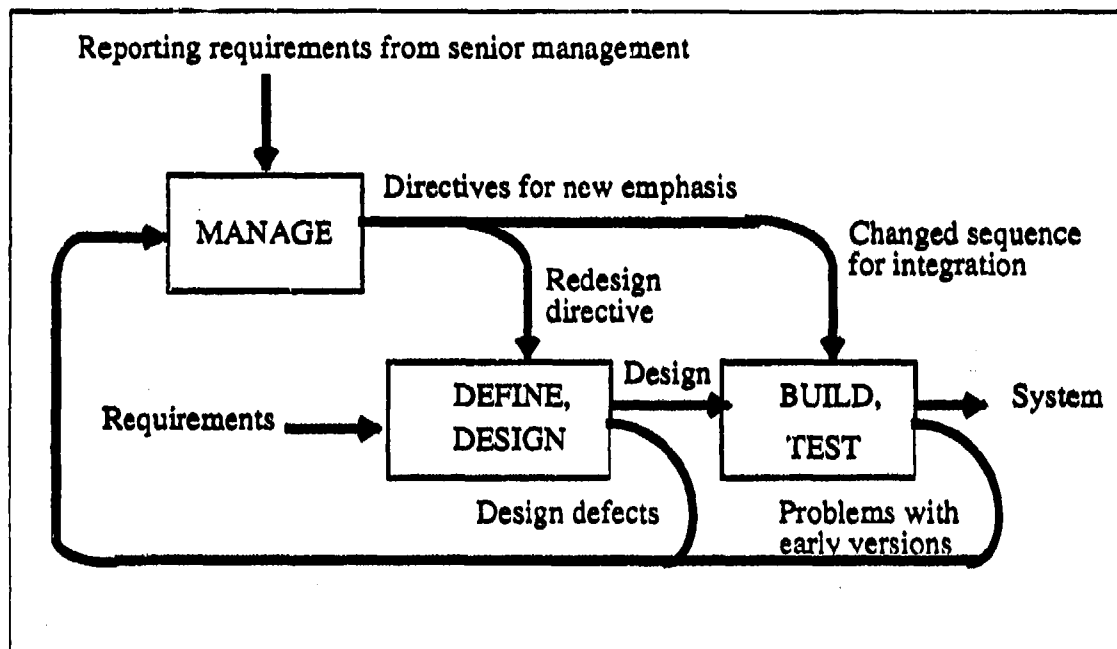


Figure 2.6 – Example of a Managed Process

The final level of process maturity, shown in Figure 2.7, is one that allows the measurements to be fed back to project management as development progresses, so that decisions about activities can be made based in part on the metrics themselves. This *optimizing* process helps to assure that course corrections can be made as an ongoing project is monitored. The optimization may involve a change in the process itself, depending on what is determined from the metrics reported during development. The dynamic tailoring of the process to the situation is indicated in Figure 2.7 by the collection of process boxes labelled T_0, T_1, \dots, T_n . At time T_0 , the process is as represented by box T_0 . However, at time T_1 (i greater than 0), management has the option of revising or changing the overall process. For example, the project manager may begin development with a standard waterfall approach. As requirements are defined and design is begun, metrics may indicate a high degree of uncertainty in the requirements. Based on this information, the process may change to one that prototypes the requirements and the design, so that the uncertainty is resolved before substantial investment is made in implementation of the current design. In this way, an optimizing process gives maximum flexibility to the development. Metrics act as sensors and monitors, and the process is not only under control but is dynamic, too.

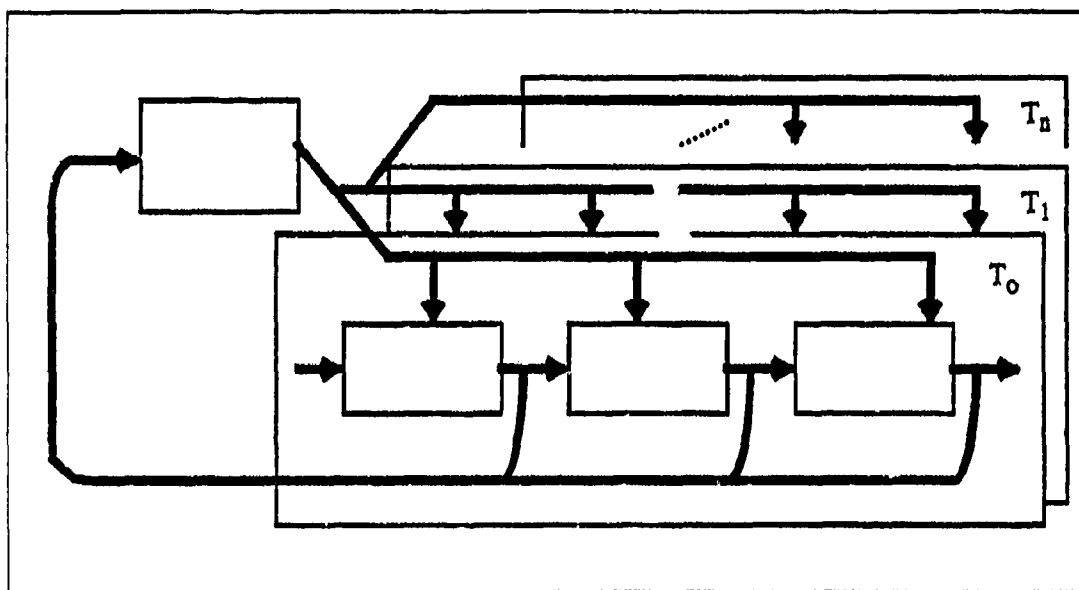


Figure 2.7 - SADT Diagram of Optimizing Process

2.3 EXPECTED BENEFITS

The process maturity framework for metrics provides several major benefits for a Contel business organization. First, the framework requires management to assess the process level for a given project or set of projects and to decide either where the software development process is or where it should be. Second, once the desired level is determined, the associated metrics offer to management the level of support needed to manage and control the development effort. The more control desired, the larger the set of metrics to be collected and analyzed. Finally, the process maturity framework acts as a set of guidelines to allow gradual movement from one level to another in a particular development organization. That is, if management decides that more control is needed, the process levels and metrics provide a vehicle for the establishment of that control and the monitoring of its effect. Likewise, improvement can be measured and documented.

The next section addresses the initial recommended set of metrics in more detail. Each phase of metrics implementation is described in terms of type of metrics, purpose for collection, expected benefits, and methods of collection and analysis.

INITIAL SET OF RECOMMENDATIONS

3.1 FORMAT OF RECOMMENDATIONS

This section describes the initial set of recommended metrics for collection at Contel. As noted in Section 2, the metrics are grouped by phase, with each phase corresponding to a process maturity level. Within a phase's metrics group, each metric is defined by the following characteristics:

- Type/class of metric: the overall class of measurements being described
- Name of candidate metric(s): particular metrics that measure the characteristic addressed by the class
- Purpose and benefit of metric: why collection and analysis of this class of data are useful
- Data required for metric calculation: what to measure and rules governing the measurement

3.2 PHASE 1: PROJECT METRICS

The project metrics correspond to a repeatable process level, although preliminary, baseline measurements may be taken at the initial process level. They include the following types of measures:

- Software size
- Personnel effort
- Requirements volatility

Each type of metric is discussed in turn.

3.2.1 TYPE: SOFTWARE SIZE

This class of metric captures some information about the amount of software being produced. It is widely acknowledged that there are several dimensions to software size, including physical size and functionality. The metrics collected in Phase 1 should include at least one of these dimensions.

Non-commented source lines of code

A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This definition specifically includes all lines containing program headers, declarations, and executable and non-executable statements. Thus, the program in Figure 3.1 has five lines of code:

LINE	
1	X := 0; Y := A^2+LOG(B)+C;
2	IF X<Y
3	THEN
4	R := X/Y
5	FI;

Figure 3.1 - Program with Five Lines of Code

This definition is the predominant one used by researchers and practitioners². Thus, its use allows easy comparison of Contel results with those of other companies and organizations. The number of lines of code (LOC) is often reported in thousands of lines of code (KLOC).

Function points

Function points were introduced at IBM³ to measure the amount of functionality inherent in a software product, rather than merely the physical size. Function points avoid the issue of variation in coding style that is often a concern of those who measure size using lines of code. The measure is calculated in three steps: computing the number of unadjusted function points (based on the inputs and outputs of the system), adjusting for complexity and calculating the final number of function points. The first step is taken by completing the chart shown in Figure 3.2.

ITEM	COUNT	WEIGHT			FP
		Simple	Average	Complex	
Number of distinct input data items	_____	3	4	6	_____
Number of output screens or reports	_____	4	5	7	_____
Number of types of on-line queries	_____	3	4	6	_____
Number of files	_____	7	10	15	_____
Number of interfaces	_____	5	7	10	_____
TOTAL FUNCTION POINTS:					

Figure 3.2 - Function Point Computation Chart

Each of the domain items specified in column one is counted and placed in the corresponding row of column two. For example, the number of interfaces counts the number of unique files or programs passed across the external boundary of the system (such as shared external utilities, math libraries, program libraries, or shared databases or files). Then, the count is multiplied by the appropriate weight from columns three, four or five and placed in column six. (The weight entries shown here were derived from the judgments of IBM researchers.) The total of the sixth column is computed for use in the second step.

Figure 3.3 displays the table used to calculate the complexity of the system. Each of the fourteen factors is rated on a scale from 0 to 5, shown at the bottom of the figure.

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily used operational environment?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Are conversion and installation included in the design?
13. Is the system designed for multiple installations in different organizations?
14. Is the application designed to facilitate change and ease of use by the user?

Ratings: 0 - no influence
 1 - incidental
 2 - moderate
 3 - average
 4 - significant
 5 - essential

Figure 3.3 - Complexity Adjustments for Function Points

The scores for each of the 14 questions are summed to form a complexity adjustment value, Q. The final step uses the following formula to compute the total number of function points for a given development project, where T is the number of unadjusted function points from the first step:

$$\text{Function Points} = T * [0.65 + .01 Q]$$

The constants .65 and .01 were determined empirically at IBM.

Object and method count

For object-oriented projects, the number of objects and methods acts as a measure of the functionality of the system. The objects and methods are counted according to the following rules:

Count all objects and methods that are explicitly defined. That is, the count includes all inherited objects and methods only if the inheritance is explicitly described in the code.

- Purpose

The software size metrics are comparative ones that are useful for determining the relative sizes of the software produced within Contel. In addition to giving us an idea of how one project compares with another, size is also a good measure for partitioning a product into categories. For example, if some code is purchased and reused, other code is reused but modified, and additional code is written from scratch, lines of code are often used as the comparative measure: 100,000 reused lines of code, 50,000 modified lines of code, and 35,000 new lines of code.

Size is often requested as a primary input to many effort and schedule estimation models. A typical estimation tool will require the user to describe certain characteristics of development: tools and techniques used, experience level of the personnel, and so on. The descriptions are used to adjust a nominal estimate of effort or schedule based on the size of the project in lines of code. Here, lines of code is a subjective estimate rather than a historical data point. Similarly, function points and object counts can be requested as input to a model or tool as an estimate of size. Historical databases of size measures are useful in this context for

suggesting and improving size estimates for future projects; this historical information helps to reduce the subjectivity inherent in any size estimate.

- Data Collection

An initial size estimate is usually made at the beginning of a project. As the project development progresses, it is helpful to re-estimate the size based on increased understanding of the project and its characteristics. By the end of the project when all code is complete, the estimate has been transformed into an actual accounting of code development. Therefore, it is recommended that estimates of size be made at regular intervals, depending on the duration of the project. For example, if a project is to last several years, size estimates should be made quarterly or monthly. Graphs of estimated versus actual size can be plotted to help management track completed code and plan for changes in budget or schedule⁴. Figure 3.4 illustrates the type of graph that can be generated from size measures.

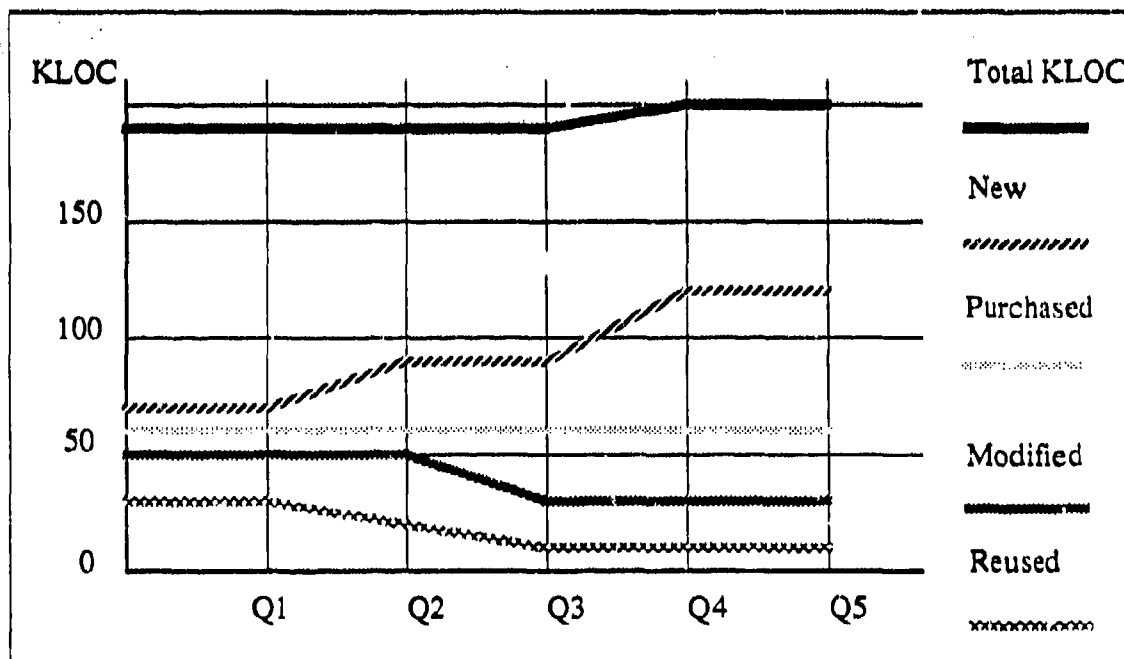


Figure 3.4 - Graph of Size Estimates Versus Actuals

Notice that the figure illustrates that size is determined in each of four categories: new code, modified code, reused code, and code purchased for this project. Size measures should be partitioned in this way for all projects at Contel. Such a division is useful in many ways. First, for systems where integration of software products is essential to the design, the tracking of categories of software in this fashion can indicate how closely the planned integration matched the actual development. Similarly, for projects where reuse plays a major role, the degree to which reused code can be used without modification can be tracked to determine how well expectations were met.

Tools are available for many environments to generate lines of code or object and method counts automatically. In situations where no tool is available, a simple one can be developed. Likewise, tools such as ASSET-R can be used to calculate the number of function points in a planned system. Many of the function point tools implement an enhanced version of function point calculation, including additional factors for augmenting the unadjusted function point count. In addition, these tools calculate a correspondence between number of function points and expected number of lines of code, depending on the implementation language. Any of these tools is acceptable at Contel; the enhanced function points may yield a more precise calculation than the original tables.

3.2.2 TYPE: PERSONNEL EFFORT

Metrics of effort must represent both the number of people working on the project and the amount of time devoted by them to the project. Two such metrics are:

Actual person-months of effort

Reported person-months of effort

These measures require knowledge of the number of people working on the project and the number of hours each one works. The number of hours is aggregated into a measure of months for ease of use.

To capture the number of people working on the project, it is important to count engineering and management staff directly involved with any of the following:

software system planning, requirements definition, design, coding, testing, documentation, configuration management and quality assurance.

Because many reporting schemes restrict the number of hours represented on a time card, it is necessary to document not only the reported hours but also the actual hours per employee.

- **Purpose**

The measures determine the amount of effort required to complete the project. As historical information, the effort data can be used to aid planners in estimating the effort needed on future projects. In addition, management of the current project can be aided by the regular collection of effort information to determine schedule adherence.

- **Data Collection**

For each person on the development team, the number of hours per month charged to the project should be reported, as well as the actual number of hours worked on the project. The total number of hours is to be reported in terms of months, and the number of months is to be multiplied by the number of employees to yield a figure in person-months. For those employees working only part-time on the project, the number of person-months should be calculated in terms of the number of full-time equivalent positions.

Hours reported can be extracted from time cards; actual hours must be gathered from development team personnel directly or through an adjunct to any reporting tool used by each team member. In both cases, hours should be reported monthly or quarterly, so that compliance with the planned effort can be determined. Figure 3.5 shows an example of the way in which data can be tracked.

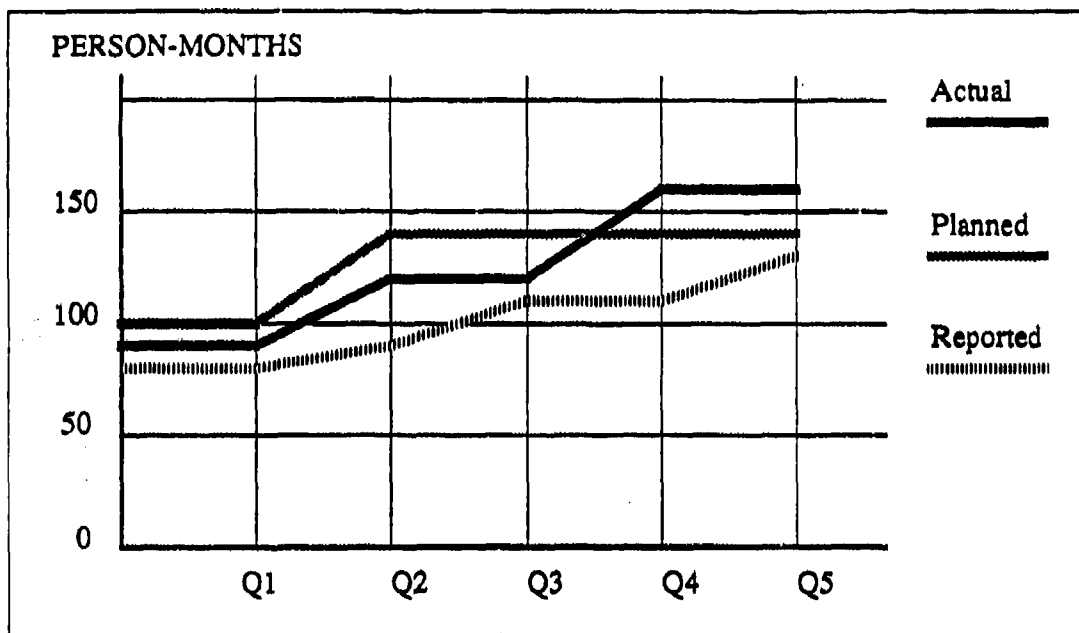


Figure 3.5 - Estimates of Planned Person-Months Versus Actual and Reported

In addition, the data can be used to determine compliance with overall schedule. As shown in Figure 3.6, the original schedule can be compared with estimates and then revised as needed. Thus, project effort measures contain within them information needed to estimate the duration of the project.

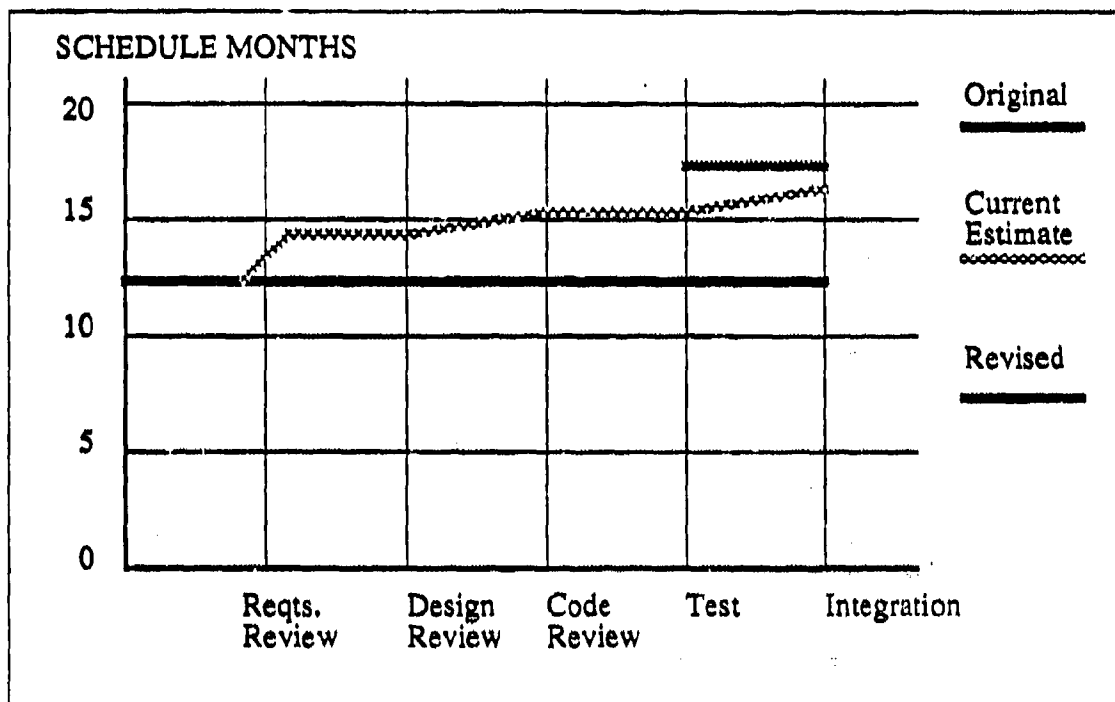


Figure 3.6 - Graph of Project Duration

3.2.3 TYPE: REQUIREMENTS VOLATILITY

Software development projects often explore new territory. Developers provide to the customer not just the automation of manual activities but the implementation of functions that can only be performed in an automated way. This exploratory venture means that the requirements evolve as development progresses. Unfortunately, the evolution of requirements can lead to problems with traceability and control. Thus, understanding the volatility of requirements is essential to project management.

Requirements changes

Requirements volatility measures the amount of change in the requirements specification as the project progresses. At a given point in time, the requirements changes metric reports the sum of the number of inserts, changes and deletions up to that time with respect to the total number of requirements for the project as listed

in the original requirements specification document. That is, the metric is calculated as:

$$(\text{number of insertions} + \text{changes} + \text{deletions}) / (\text{number of requirements})$$

- **Purpose**

For many projects, requirements volatility is a major factor in causing a project to exceed its budget or schedule. Changes to the requirements are expected in the early stages of design as the details of the system's operation are being understood by the development team. However, at some point, the number of changes should level off. Thus, the tracking of changes to requirements can be a useful aid in keeping management abreast of the stability of the requirements and the adequacy of effort and schedule estimates.

- **Data Collection**

The total number of requirement items must be counted. This measure must be made in accordance with whatever accounting standard is used for reporting and configuration management purposes. For example, some customers require that computer software components be tracked against requirements, and the method for enumerating requirements is part of the contract. Other development efforts segregate requirements by the appearance of the word "shall", with each "shall" corresponding to one requirement. Changes to the requirements include the insertion or deletion of text as well as modification of existing text.

The original requirements specification will be used as the baseline document against which all changes are measured. Both the total number of requirements and the number of changes to requirements will be measured monthly or quarterly, depending on what is appropriate for the overall duration of development. The measurement can be made manually or with the help of configuration management tools.

Figure 3.7 illustrates the tracking of changes against total requirements. It reflects the fact that the total number of requirements is likely to grow as the customer's problem is explored by the development team. The horizontal axis can track not

only the passing of time but also the significant milestones of the development process. A steady increase in changes to requirements after the final design is approved may signal a problem in understanding and the likelihood of resulting schedule and budget changes.

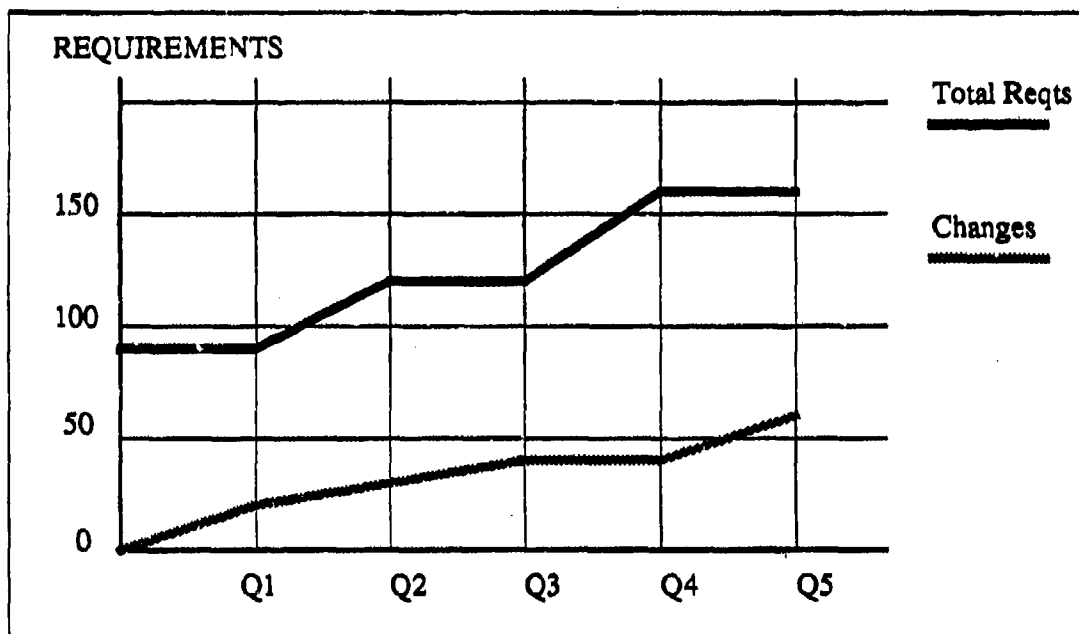


Figure 3.7 - Graph of Requirements and Changes

3.2.4 POSSIBLE ADDITIONAL PROJECT MANAGEMENT METRICS

Several additional metrics may be desirable, depending on the characteristics of the project and the needs of project management. Many studies of project cost indicate that experience and employee turnover can have a significant impact on overall project cost. Thus, the following items can be added to the Phase 1 metrics set at the discretion of management.

- Experience
 - with domain/application
 - with development architecture
 - with tools/methods

-- overall years of experience

The experience is measured in terms of number of years of experience. It is sometimes useful to graph the experience data with productivity data to determine if experience and productivity are correlated. It is helpful to note the range of productivity for a given level of experience, as the research literature reports as much as a factor of ten difference in productivity for equivalent experience. Figure 3.8 illustrates this result.

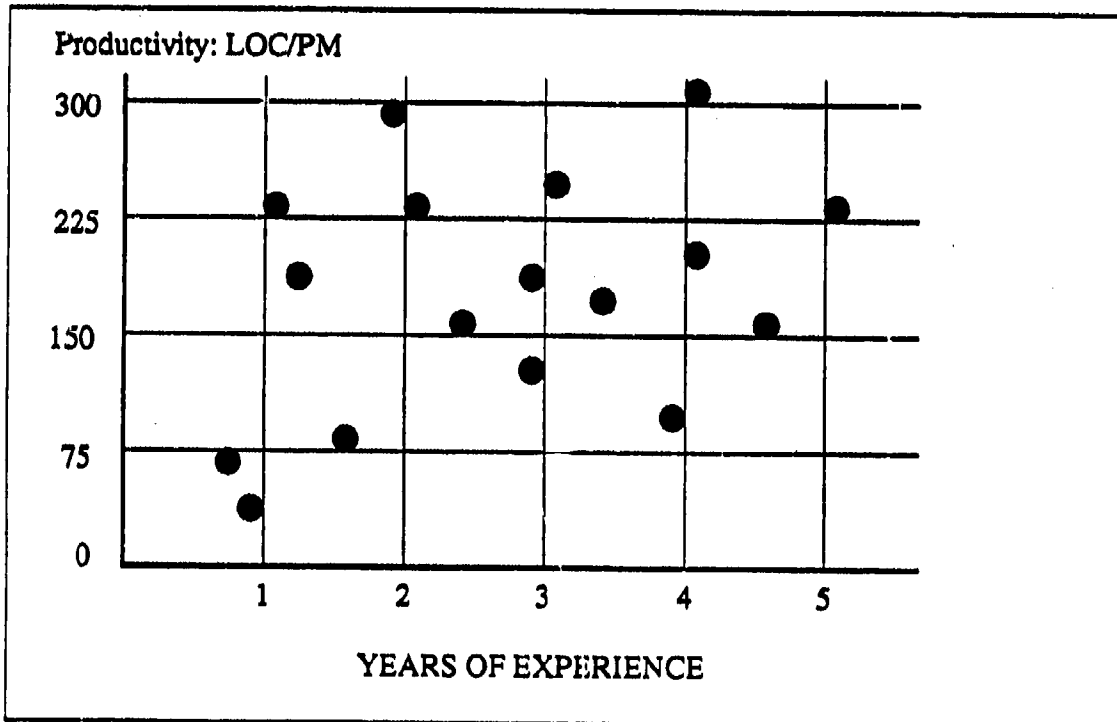


Figure 3.8 - Graph of Experience vs. Productivity

- Employee turnover

Employee turnover can be measured in terms of the number or percentage of employees leaving the project. Many managers consider turnover to be the primary reason for failure of projects to meet schedule deadlines. Information about the employees working on a development project can be clustered by level of experience, using the classes described above. Turnover can be compared in a graph with productivity or schedule to determine whether turnover appears to have

an effect on them. Such information may be useful in assessing the impact of turnover on existing projects and on estimating turnover effects on future ones.

3.3 PHASE 2: PRODUCT METRICS

Regardless of the actual development process employed, being at the level of a defined process means that one activity is separable from another, and distinct products are produced. Thus, the recommended metrics for the second phase require a defined process and address the complexity and quality of the products of each activity. In terms of complexity, it is suggested that the following items be examined:

- Requirements complexity
- Design complexity
- Code complexity
- Test complexity

In some cases, complexity also can be considered as a measure of product quality. For example, code that is less complex is of higher quality than code that is more complex. Thus, tracking the complexity of intermediate software products can indicate the likely quality of the code.

Another perspective from which to view the quality of the products is examination of the number of faults in each product and the density of defects overall. In addition, the thoroughness of testing can be assessed. Thus, quality metrics include:

- Defects discovered
- Defects discovered per unit size (defect density)
- Requirements faults discovered
- Design faults discovered
- Code faults discovered

It is important to note that this set does not represent the full spectrum of quality measures that can be employed. Issues of maintainability, utility, ease of use and other aspects of quality software are not addressed by defect counts. However, defect analysis is relatively easy to implement, and it provides a wide spectrum of useful information about the reliability of the software and the thoroughness of testing.

3.3.1 TYPE: REQUIREMENTS COMPLEXITY

The complexity of requirements plays a large role in determining the difficulty involved in building the system. However, measuring the complexity of requirements is compounded by the variety of ways in which requirements are recorded and tracked. The metric suggested here, although a simple measure of size, also captures the major components of complexity.

Number of distinct objects and actions addressed in requirements

For each requirement specified, a count is made of the distinct objects to be manipulated by the system and of the actions that can be performed on them or taken by them. For example, the set of requirements in Figure 3.9 contains six objects, each of which is bolded, and five actions, each of which is italicized. (Bolding and italics are shown only when the object or action is first encountered, so there is no double-counting.)

The system shall provide the capability to *update* the directory when *creating* or *modifying* a Telephone Number Record (TNR).

The system shall provide the date of last update for each listing.

The system shall automatically update the directory when modifications are made to **directory fields** in TNRs.

The system shall provide a **Directory Activity Log**, which shall be automatically *updated* when directory-associated creations, *deletions* and changes occur to TNRs.

Figure 3.9 - Example Requirements with Objects and Actions

Notice that only distinct objects and actions are counted. Moreover, an action is always associated with an object. Therefore, "update" is counted twice: once for a directory update, and once for a directory log update.

- **Purpose**

The number of distinct objects and actions is used as an indicator of the relative complexity of the requirements. That is, one set of requirements is considered to be more complex than another if the first has more objects and actions. Such a measure can be used in two ways. First, the complexity of the requirements can be tracked over the duration of the project. As the requirements change, an increase in complexity can be viewed as a warning that the estimates of schedule and effort may need revision. Second, the complexity of the requirements can be viewed in concert with the complexity of the design and code. Based on historical data, the complexity of requirements can be used as a predictor of the expected complexity of the subsequent products. Disparities may be indicators of problems inherent in the design or code that do not necessarily follow from the complexity of requirements.

- **Data Collection**

The count of objects and actions may be automated and incorporated in whatever software tracks the changes to requirements. As with requirements changes, graphs may be generated to follow the growth in complexity over time, as shown in Figure 3.10.

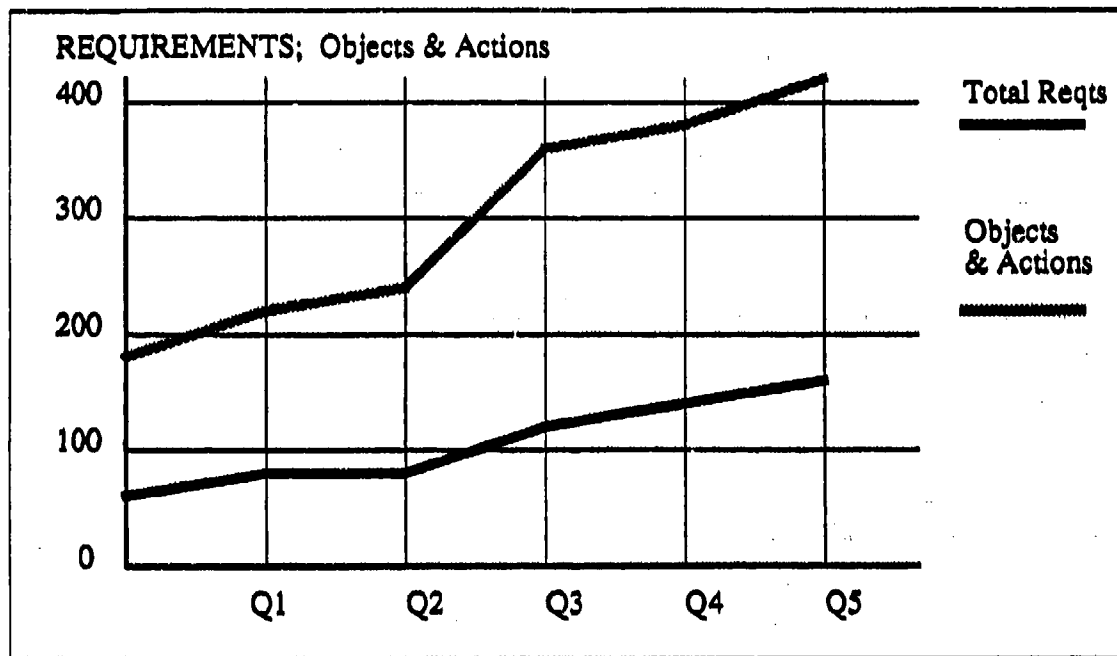


Figure 3.10 - Graph of Requirements and Complexity

3.3.2 TYPE: DESIGN COMPLEXITY

Measures of design complexity capture the aspects of design that make it difficult to comprehend or implement. Again, the variety of methods for documenting a system's design makes it difficult to choose a universal measure to analyze design complexity. The measures suggested below are simple size measures but count the design components that are responsible for contributing substantial complexity to the design.

Number of design modules

When a large number of modules is included in a design, a large number of interfaces contributes to the design's complexity. The design can be represented in a number of ways, both textually and graphically. These representations may include program design language, SADT diagrams, data structure diagrams, and others. When the system involves a great deal of data or even a database management system, the design may be documented in terms of entities and relationships. Therefore, a count of the number of design modules must be tailored

to the design representation scheme. The count may be the number of entity-relationship components, the number of program design language modules, or something similar. The overriding factors in deciding what to count include:

- Assurance that the count can be made repeatedly over the life of the system being developed
- Capture of a measure that increases along with both the number of objects being manipulated by the system and the number of functions performed by and on those objects

Cyclomatic complexity

The cyclomatic complexity number is based on the number of decisions embedded in the design. It was originally aimed at measuring the number of linearly independent paths through a program, and it is based on the graph formed by a flow chart of the design. If the flow chart has e edges and n nodes, then the cyclomatic complexity of the design represented by the graph is

$$e - n + 2$$

For example, Figure 3.11 is the graph of a design with 13 nodes and 16 edges, so its cyclomatic complexity is 5.

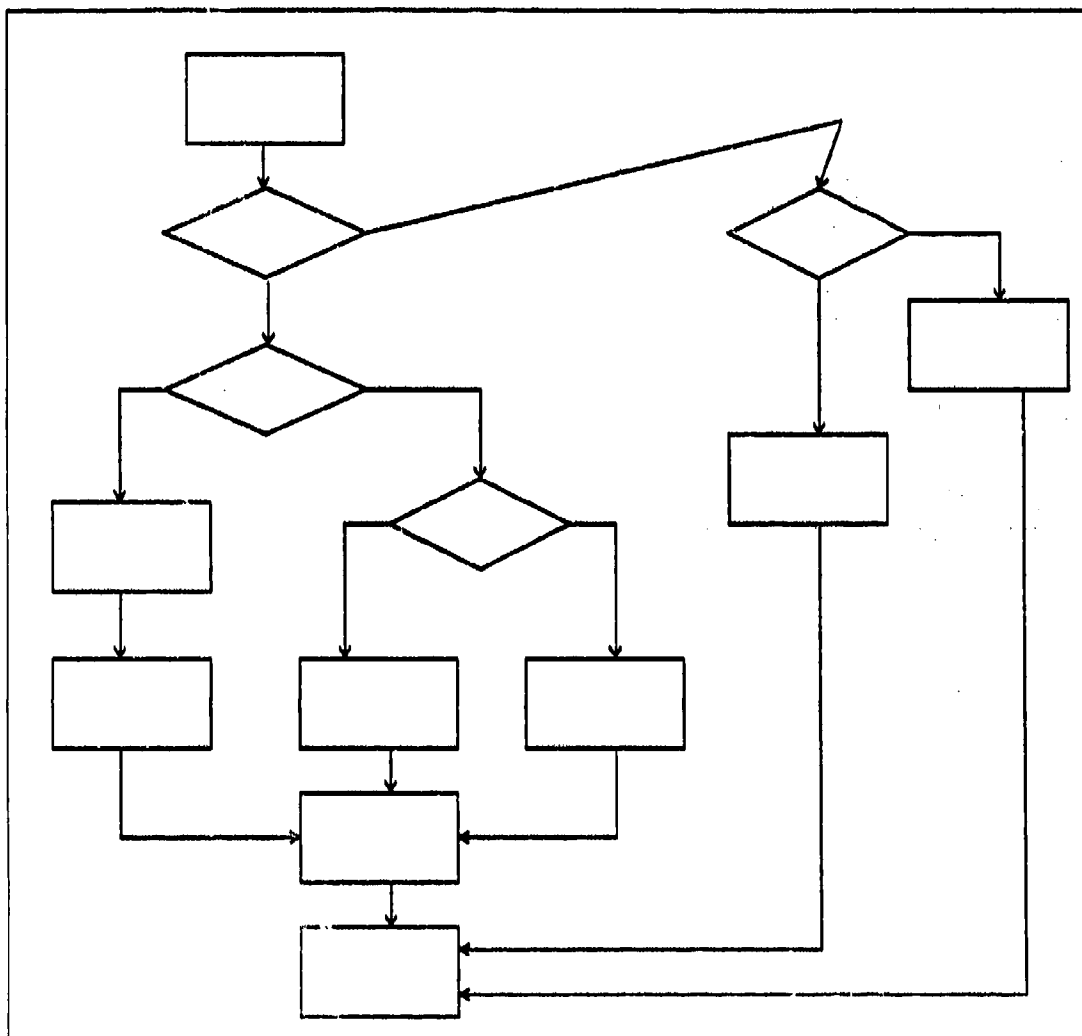


Figure 3.11 - Graph with Complexity = 5

- Purpose

The measures of design complexity allow management to determine whether the complexity of the design changes as development progresses. For example, if requirements change, the impact of the changes on the design can be assessed. Furthermore, the complexity of both the requirements and the design can be considered when predicting the likely complexity of the resulting code.

- Data Collection

Both the count of design modules and the calculation of cyclomatic complexity can be performed manually. However, automated tools make the job much easier; and they are available for both calculations. Many of the automated design tools include the ability to count the number of modules or entities and relationships. The cyclomatic complexity metric is popular, and its computation has been implemented by a variety of software vendors.

The design complexity data can be viewed graphically in conjunction with other metrics to serve as an indicator of the overall complexity of the problem being solved. For example, the graph shown in Figure 3.12 compares the growth of the number of requirements with the number of design modules. In this example, the system appears to be growing more complex as requirements are added.

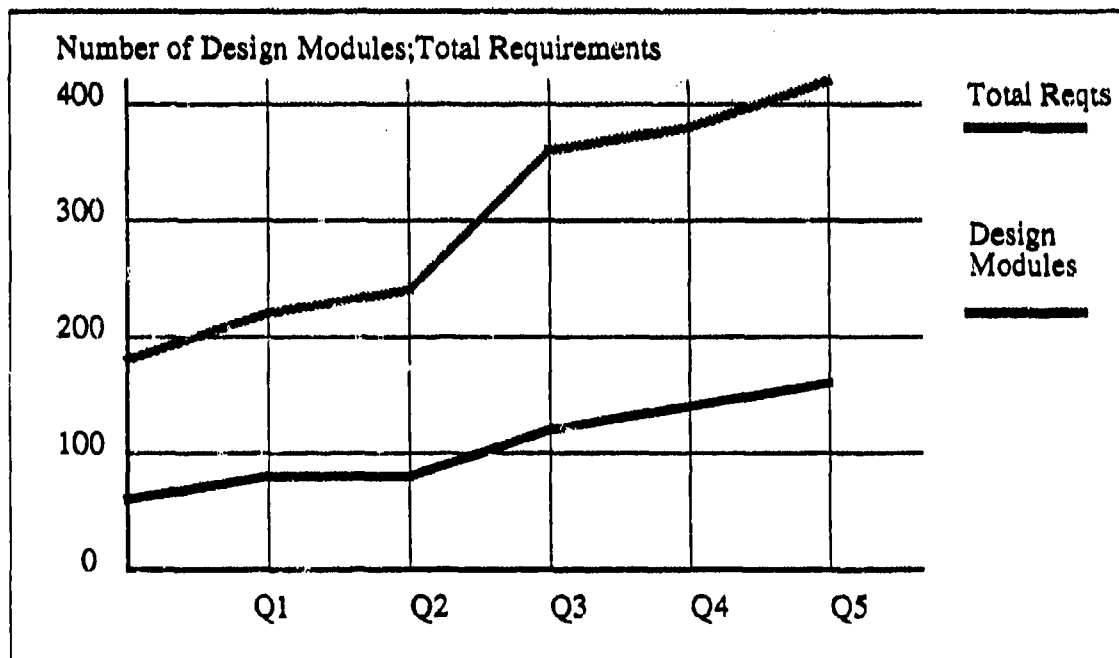


Figure 3.12 - Graph of Design Complexity Compared with Total Requirements

3.3.3 TYPE: CODE COMPLEXITY

As with measures of requirements and design complexity, measures of code complexity capture the aspects of the implementation that make the structure of the code difficult to comprehend or change. These measures include the following:

Number of code modules

When a large number of modules is included in a program, a large number of interfaces contributes to the program's complexity. The definition of "module" depends on the implementation language chosen. However defined, a module must be easily recognizable, so that two independent people will generate the same count from the same set of programs.

Cyclomatic complexity

The cyclomatic complexity number is based on the number of decisions embedded in the program. As with the design, the cyclomatic complexity of a program is generated from the structure formed by a flow chart of the program. If the flow chart has e edges and n nodes, then the cyclomatic complexity of the program represented by the graph is

$$e - n + 2$$

For example, Figure 3.13 is the graph of a program with 13 nodes and 16 edges, so its cyclomatic complexity is 5.

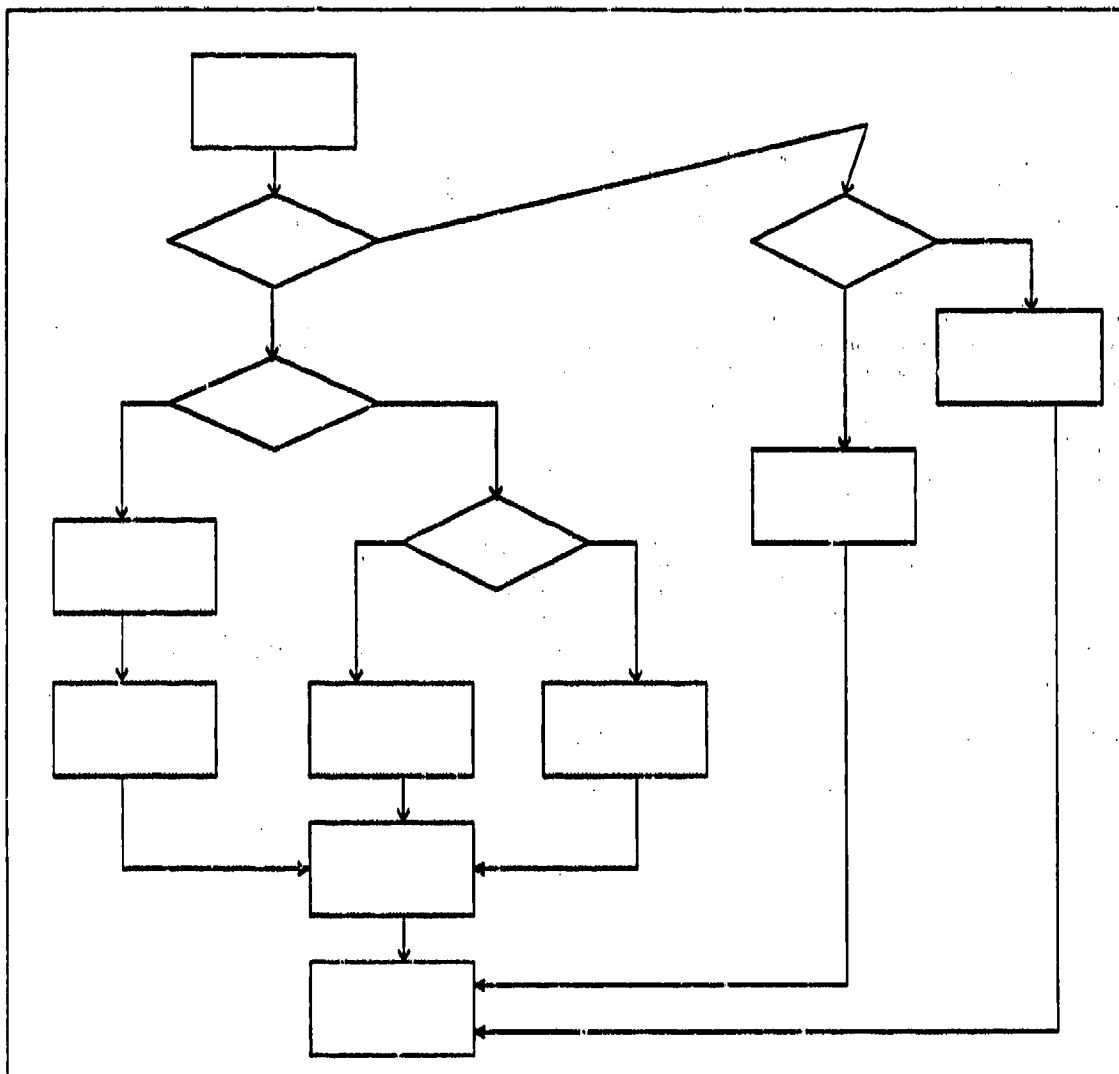


Figure 3.13 - Graph with Complexity = 5

- **Purpose**

The measures of code complexity allow management to determine whether the complexity of the programs changes as development progresses. For example, if requirements change, the impact of the changes on the code can be assessed. Furthermore, the complexity of both the requirements and the code can be tracked to determine the relationship between the two.

- Data Collection

Both the count of modules and the calculation of cyclomatic complexity can be performed manually. However, automated tools make the job much easier, and they are available for both calculations. Many programmer tools include the ability to count the number of modules. The cyclomatic complexity metric is popular, and its computation has been implemented by a variety of software vendors.

Code complexity data can be viewed graphically in conjunction with other metrics to serve as an indicator of the overall complexity of the problem being solved. For example, the graph shown in Figure 3.14 compares the growth of the number of requirements with the number of code modules. In this example, the system appears to be growing more complex as requirements are added.

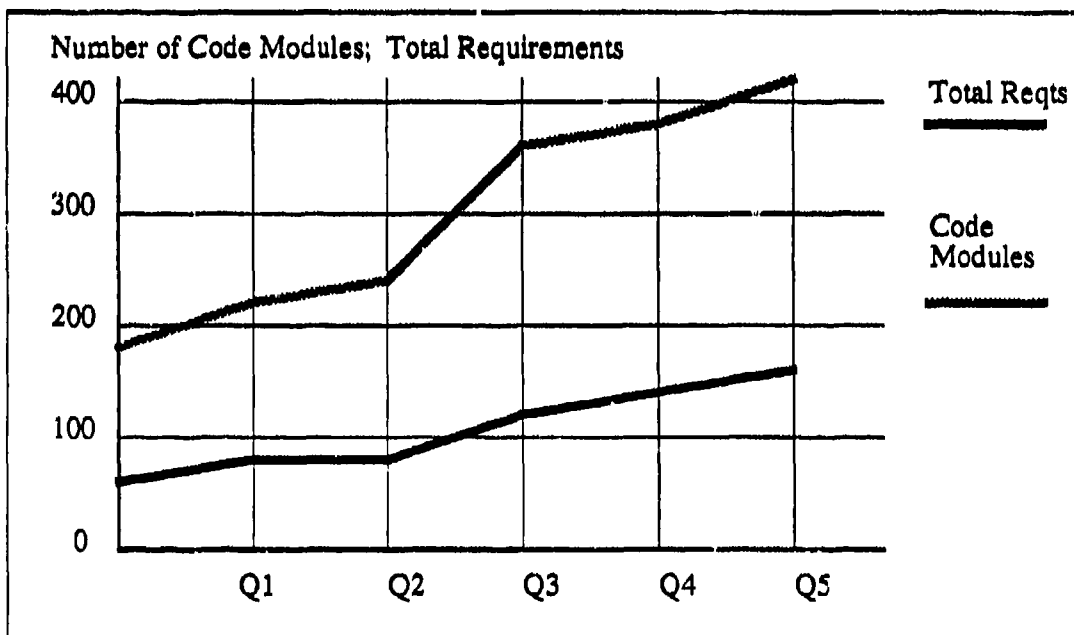


Figure 3.14 - Graph of Code Complexity Compared with Total Requirements

McCabe, the creator of the cyclomatic complexity measure, suggests that the complexity of a module be held at 10 or less. There is little conclusive evidence in the literature to support the reasonableness of this cut-off. However, each development organization can establish guidelines for complexity based on past experience. For example, the graph illustrated in Figure 3.15 shows that the average

complexity has been lowered while the number of code modules has increased slightly.

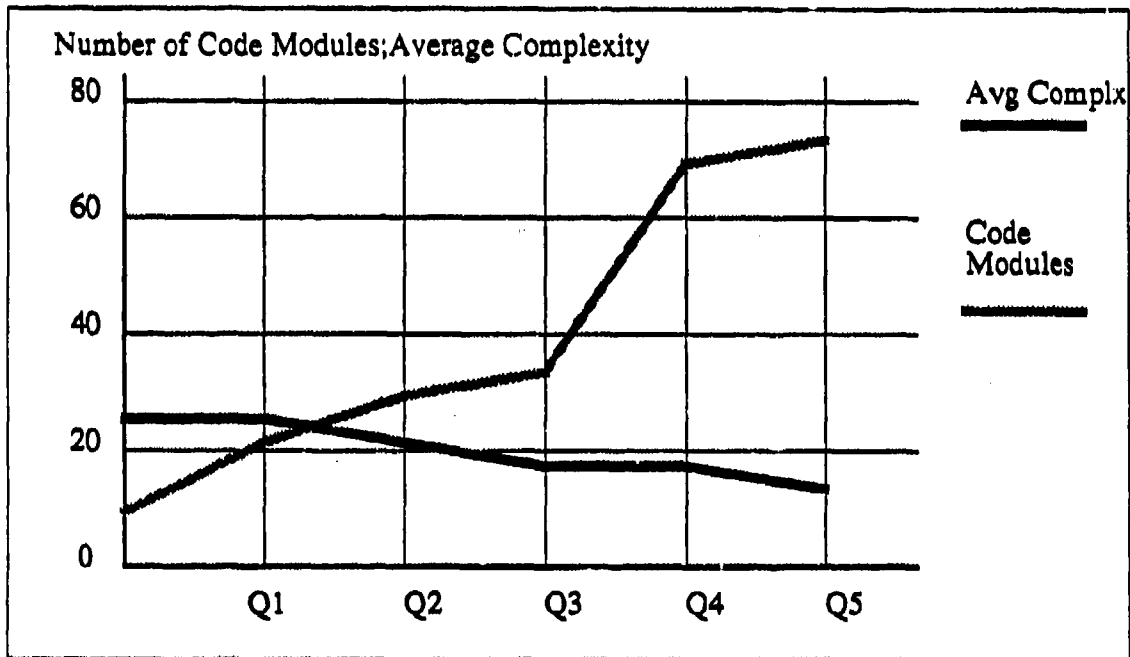


Figure 3.15 - Graph of Code Complexity Compared with Total Requirements

3.3.4 TYPE: TEST COMPLEXITY

The determination of test complexity depends to some extent on the implementation of the system. For example, if the system is implemented using a language or technique that can easily be viewed as a flow chart, then examination of the number of paths to test is feasible. If, however, the implementation is object-oriented, then the number of object interfaces may be a more appropriate indicator of test complexity.

Number of paths to test

Each module in a system can be evaluated in terms of the number of paths that must be tested. For example, the program in Figure 3.16 can be viewed as the flow chart of Figure 3.17.

```

Line
1  SUBROUTINE SORT (X,N)
2  INTEGER X(100), N, I, J, SAVE, IM1
3  C THIS ROUTINE SORTS ARRAY X INTO ASCENDING
4  C ORDER
5  IF(N.LT.2) GO TO 200
6      DO 210 I=2,N
7          IM1=I-1
8          DO 220 J=1,IM1
9              IF(X(I).GE.X(J)) GO TO 220
10             SAVE=X(I)
11             X(I)=X(J)
12             X(J)=SAVE
13 220      CONTINUE
14 210      CONTINUE
15 200      RETURN
16  END

```

Figure 3.16 - Sample FORTRAN Program

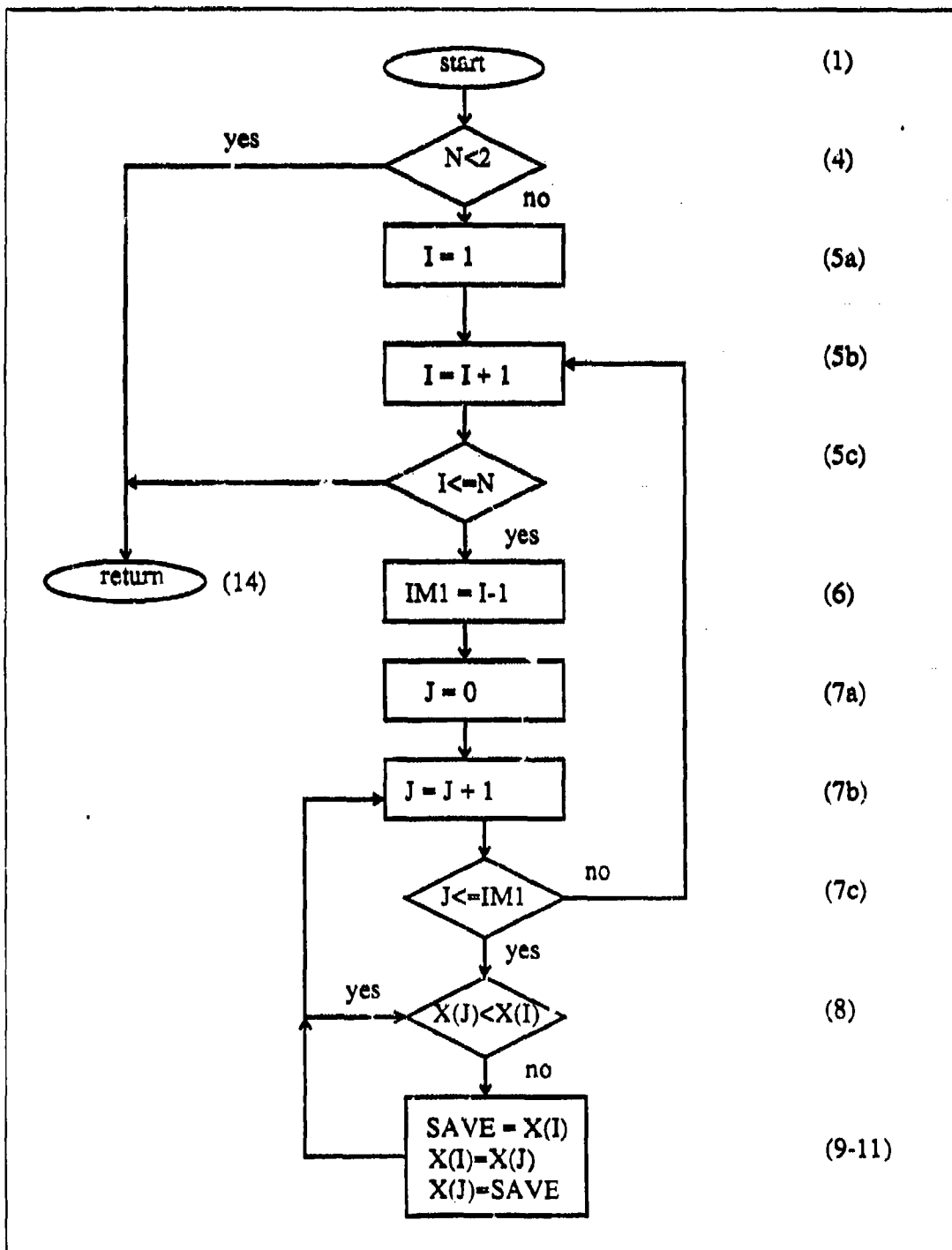


Figure 3.17 - Flow Chart of FORTRAN Program

In this example, there are five unique paths from start to finish, so there are five paths to test:

1-4-14

1-4-5a-5b-5c-14

1-4-5a-5b-5c-6-7a-7b-7c-5b-5c-14

1-4-5a-5b-5c-6-7a-7b-7c-8-7b-7c-5b-5c-14

1-4-5a-5b-5c-6-7a-7b-7c-8-9thru11-7b-7c-5b-5c-14

If object-oriented development, number of object interfaces to test

For object-oriented development, the number of object interfaces is determined by the number of actions that can be taken by an object and done to an object. Thus, the number of object interfaces is the sum of all methods for an object. If two different objects can exercise the same method, then that method should be tested (and counted) twice.

- Purpose

The test complexity measure serves two functions. First, it can be compared with the complexity of the requirements, design and code to determine whether a correlation exists. If so, the complexity of an earlier product, such as requirements, can be used to predict the likely complexity of testing. Such information can be useful in planning the amount of time needed for thorough testing.

Second, the test complexity measure can be used to determine completeness and sufficiency of testing. The number of paths or interfaces actually tested can be divided by the total number of paths or interfaces (that is, the test complexity) to compute a measure of sufficiency.

- Data Collection

The test complexity information for a set of projects can be graphed with other complexity data to explore the relationships between the variables. A typical graph

comparing requirements complexity with average test complexity per module is shown in Figure 3.18. The dashed line is the result of a linear regression analysis; it suggests that an increase in the complexity of the requirements is likely to yield an increase in test complexity.

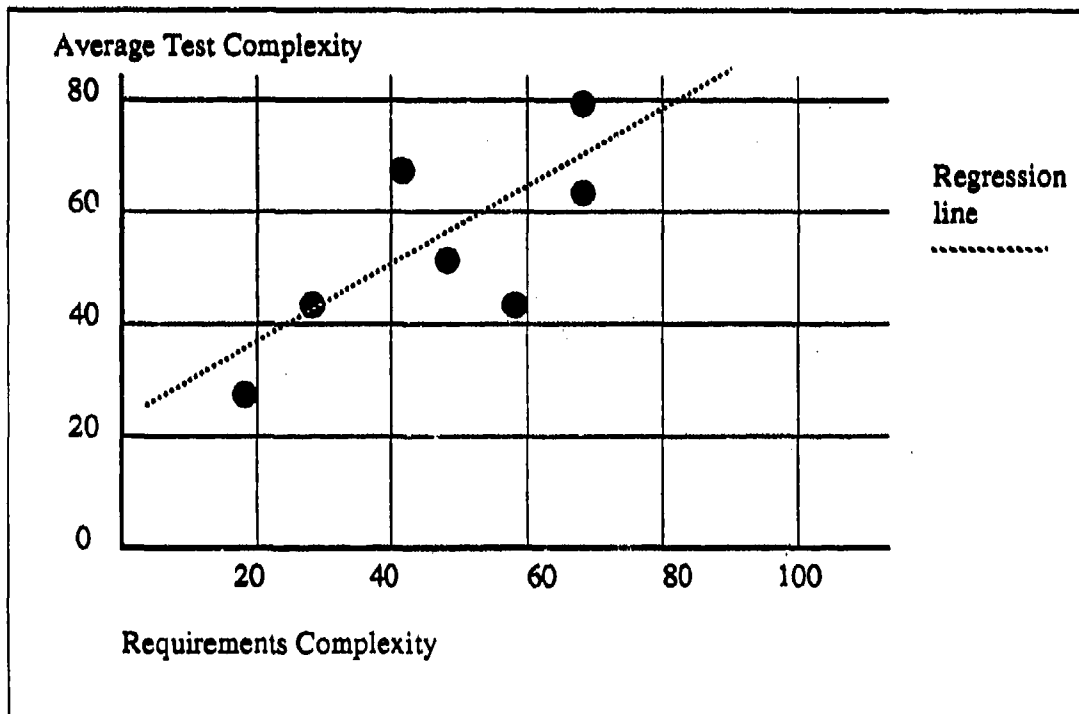


Figure 3.18 - Graph of Test Complexity Compared with Requirements Complexity

The test sufficiency also can be calculated, as described above. The sufficiency data for a set of projects can be compared with information about acceptance testing, number of defects or maintenance effort to determine whether sufficiency can be used as a predictive tool.

In many instances, automated tools are available to calculate the number of paths directly from the code. These tools depend on the implementation language. More information about appropriate tools can be obtained from the Process and Metrics Project.

3.3.5 TYPE: DEFECTS DISCOVERED

Before measures of defects can be described, it is important to understand the difference between a fault and a defect. A fault is a software error that causes the software to produce an incorrect result for a valid input. A defect is evidence of the existence of a fault. Thus, an input set can reveal several different faults.

The documentation of defects usually begins during reviews of the requirements and continues throughout the development process. Defects can be identified at walkthroughs, reviews or inspections of each product, as well as during formal testing. The number of defects can be useful in evaluating the quality of the resulting system.

Count of defects

The actual number of defects must be determined dynamically. Once an error causing a given defect is detected and fixed, the system should be reevaluated to determine whether the same input set still yields an incorrect result. If yes, then either there originally were at least two defects, or the change to fix the first defect introduced a second defect.

The number of defects should be counted with respect to the severity of the defect. A severity scale from 1 to 4 can be defined for most projects:

1. Minor local impact: Defect causes annoyance or incorrect result; system functioning is not impeded; effect is isolated.
2. Minor global impact: Defect causes annoyance or incorrect result; system functioning is not impeded; effect is widespread and affects multiple functions.
3. Major local impact: Defect prevents the exercise of a major system function; other system functions can be exercised despite defect.
4. Major global impact: Defect prevents the exercise of at least one major system function; other system functions impaired or inoperable because of defect.

- **Purpose**

The number of defects in a system is a broad indication of the degree to which the system meets its requirements. Moreover, each defect must be tracked, its cause determined and a solution or workaround implemented. In this way, the number of corrected defects is a measure of the effectiveness of testing.

- **Data Collection**

The number of defects can be tracked with the configuration management system and test reporting tools associated with development. If such a system or tools do not exist, defects must be documented manually.

It is useful to view defects from several perspectives. First, the number of defects discovered can be reported weekly, monthly and as a total figure. As testing progresses, the number of defects discovered each week should decrease, as shown in Figure 3.19; otherwise, it is likely that correction of existing defects is introducing new faults, or that a major flaw exists in the design.

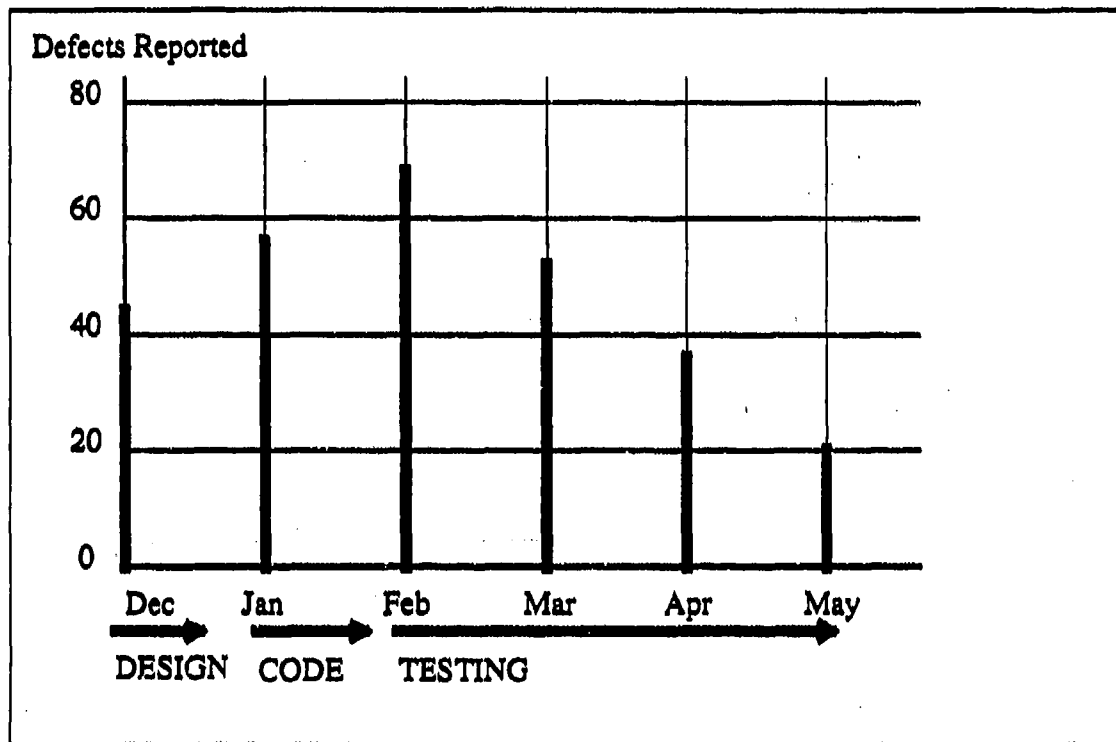


Figure 3.19 - Graph of Defects Discovered Over Time

It is also helpful to measure the number of defects per unit of size, where size is measured as described in section 3.2.1. Such a measure is called the *defect density*, and is used as an overall measure of the quality of the finished product. Historical information about defect density can be evaluated with respect to several process characteristics to determine whether certain techniques or tools have an effect upon the defect density.

Another useful indicator of quality is the percentage of defects reported that have been corrected. Ideally, all defects should be eliminated, but it is often impossible to do so. The tracking over time of the percentage of defects eliminated is a measure of the speed at which the development team responds to defect reports as well as an indicator of how close the product is to its quality goals. Figure 3.20 depicts a graph tracking percentage of corrected defects.

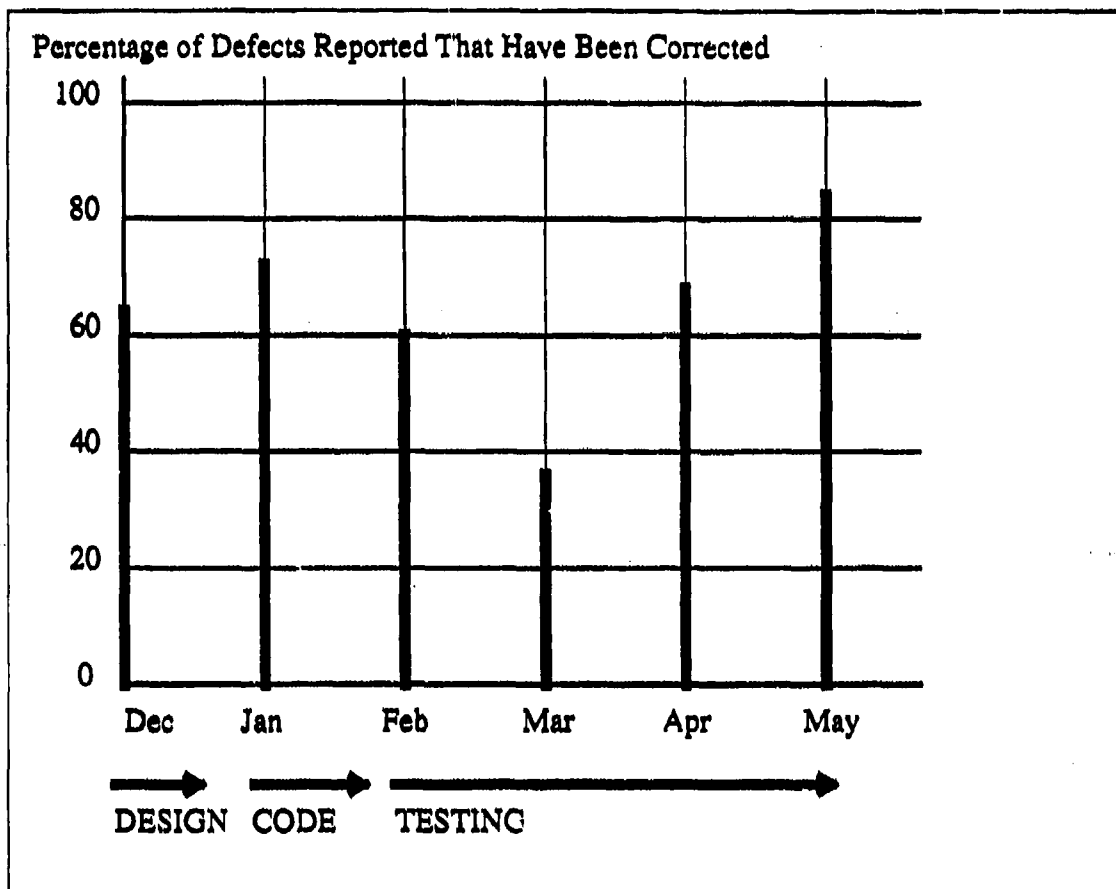


Figure 3.20 - Graph of Percentage of Defects Corrected Over Time

3.3.6 TYPE: REQUIREMENTS FAULTS DISCOVERED

Once defects are reported, the faults causing those defects must be identified, analyzed and corrected. It is important to know where the faults occur, because the number of faults in each product is an indication of the quality of that product. Each time a defect is counted, the corresponding faults and their locations should be tracked. Faults in requirements include ambiguous terminology, conflicting requirements or incomplete specifications.

Count of faults found in the requirements

The count of faults in requirements is made as each defect is corrected. If one fault causes more than one defect, the fault is counted only once.

- **Purpose**

The number of faults in requirements acts as a measure of the quality of the requirements.

- **Data Collection**

The number of faults in requirements should be collected as defects are tracked and corrected. It may be useful to calculate the number of faults as a percentage of the total number of requirements to yield an overall measure of quality. Ten faults out of 20 requirements means that the requirements specification is of poor quality, whereas 10 faults out of 1500 requirements indicates a relatively good set of initial requirements.

3.3.7 TYPE: DESIGN FAULTS DISCOVERED

Similarly, the faults in design that are identified as causing defects must be analyzed and corrected. Faults in design include ambiguous terminology, conflicting or incomplete definitions, conflict with the requirements, lack of cohesion, inappropriate coupling, or improper documentation.

Count of faults found in the design

The count of faults in design is made as each defect is corrected. If one fault causes more than one defect, the fault is counted only once.

- **Purpose**

The number of faults in design acts as a measure of the quality of the design.

- **Data Collection**

The number of faults in design should be collected as defects are tracked and corrected. It may be useful to calculate the number of faults as a percentage of the total number of design components or modules to yield an overall measure of quality. Ten faults out of 20 design components means that the design is of poor quality, whereas 10 faults out of 1500 components indicates a relatively good design.

3.3.8 TYPE: CODE FAULTS DISCOVERED

Finally, the faults in the code that are identified as causing defects must be analyzed and corrected. Faults in code include conflicting or incomplete definitions, conflict with the requirements or design, misuse of a language construct, improper syntax, lack of cohesion, inappropriate coupling or improper documentation.

Count of faults found in the code

The count of faults in the code is made as each defect is corrected. If one fault causes more than one defect, the fault is counted only once.

- **Purpose**

The number of faults in the code acts as a measure of the quality of the code.

- **Data Collection**

The number of faults in code should be collected as defects are tracked and corrected. It may be useful to calculate the number of faults as a percentage of the total number of code components or modules, or as a percentage of the software size to yield an overall measure of quality. Ten faults out of 20 code modules means that the design is of poor quality, whereas 10 faults out of 1500 modules indicates a relatively good implementation. Similarly, 10 faults in 200 lines of code is not as good as 10 faults in 200,000 lines of code.

3.3.9 POSSIBLE ADDITIONAL PRODUCT METRICS

When customer requirements dictate that significant amounts of documentation be written (as often happens on government contracts), the number of pages of documentation may be a desirable characteristic to track and correlate with effort or duration. The set of product metrics also may include number of pages of documentation as a product metric.

3.4 PHASE 3: PROCESS METRICS

The preceding sections have addressed the first three levels of process maturity and the corresponding appropriate metrics. Studies by the Software Engineering Institute of more than 100 software development organizations reveal that most projects are at levels 1, 2 and

3. Very few development projects have reached levels 4 or 5: the managed or optimizing levels. For this reason, this section discusses only the types of metrics recommended at level 4, but does not provide the detail of previous sections.

If the maturity of the process has reached the managed level, then process-wide metrics can be collected and analyzed. These metrics reflect characteristics of the overall process and of the interaction among components of the process. A distinguishing characteristic of a managed process is that the development of software can be carefully controlled. Thus, a major characteristic of the recommended metrics is that they help management to control the development process.

The Process and Metrics Project recommends that the following types of process metrics be considered for a managed project:

- **Process type:** What process model is used in development? For example, the waterfall, prototype and transformational development paradigms are very different. Examination of the process type, in concert with other product and process characteristics, may reveal that the type of process correlates highly with certain positive or negative consequences.
- **Amount of producer reuse:** How much of what products is designed for reuse? This measure includes reuse of requirements, design modules and test plans as well as code. By designing components for reuse, one project group may benefit from the effort of another group. Effort in understanding, creating and testing code can be minimized, thus making the project easier to control. Furthermore, future projects can benefit from the reuse of components produced here.
- **Amount of consumer reuse:** How much of the project reuses components from other projects? This measure includes reuse of requirements, design modules and test plans as well as code. By using tested, proven components, effort can be minimized and quality can be improved.
- **Defect identification:** How and when are defects discovered? Knowing whether defects are discovered during requirements reviews, design reviews, code walkthrough and reviews, integration testing or acceptance testing will indicate whether those process activities are effective.

- **Use of defect density model for testing:** To what extent does the number of defects determine when testing is complete? Many organizations have no overall defect goals for testing the product, so there is no way to judge the quality of either the testing or the code. The use of defect density models has been shown to control and focus testing as well as to increase the quality of the final product.
- **Use of configuration management:** Is a configuration management scheme imposed on the development process? Configuration management and change control afford management a great deal of control over the development process. Traceability links can be used to assess the impact of alterations in some or all development activities or products.
- **Module completion over time:** At what rates are modules being completed? Although ostensibly a product metric, the rate at which modules are identified, designed, coded, tested and integrated reflects the degree to which the process and development environment facilitate implementation and testing. If the rate is slow, the process may need improvement.
- **Capital intensity:** To what degree do capital expenditures affect the budget and productivity of a project? The software engineering marketplace is rife with tools and techniques whose proponents claim will increase the productivity of development teams and the quality of products. Measures of capital intensity allow measurement of the actual effects of capital investment on quality and productivity. These historical metrics, evaluated during a post mortem of a project, tell us whether we have benefited from our investment and to what extent. Such information is useful in planning new projects. In particular, measures of capital intensity can be used to decide the number of projects over which a capital investment must be amortized in order to reap its benefits.

All of the process metrics described above are to be used in concert with the metrics discussed in earlier sections. Relationships can be determined between product characteristics and process variables to assess whether certain processes or aspects of the process are effective at meeting productivity or quality goals. The list of process measures is by no means complete and is suggested only as an initial attempt to capture important information about the process itself.

3.5 PHASE 4: PROCESS METRICS WITH FEEDBACK FOR CHANGE

The final phase of metrics collection involves an optimizing process. Here, feedback allows not only control of process activities but also the possibility of changing the process itself as development progresses. This process phase can follow only from a well-controlled process (that is, a managed process). Its associated metrics will not be recommended until the metrics for Phases 1, 2 and 3 are well-established on Contel projects and evaluated for their utility and practicality.

The next section of this document explains how Contel business units can begin to establish a comprehensive metrics program based on the recommended metrics.

SECTION 4

USING THE METRICS SETS

4.1 STEPS TO TAKE IN USING THE METRICS SETS

The recommended metrics of the previous section are most useful when implemented in a careful sequence of process and metrics activities. These activities lay the groundwork for effective project management by evaluating the needs and characteristics of development before the appropriate phase of metrics collection is identified. The typical Contel business unit should take the following steps:

- **Conduct process assessment:** Working with a set of guidelines or with a member of the Process and Metrics team, the project management should determine the level of process maturity achievable (for a proposed project) or implemented (for an on-going one).
- **Consult with Process and Metrics Project team:** Once the process level is known, the Process and Metrics team and the project managers can decide which phase of metrics collection is most appropriate. For example, if Phase 2 is indicated by the process level, but the on-going project currently collects no metrics data at all, then Phase 1 may be suggested as a starting point, with Phase 2 metrics to be added at a later time. The result of the consultation will be an action plan that describes which metrics to collect and when to collect them.
- **Recommend metrics, tools, techniques:** When the types of metrics are determined, the project managers should identify tools and techniques to be used on the project. These tools and techniques are to be chosen with the overall goals of the project in mind. Whenever possible, automated support for metrics collection and analysis will be implemented as part of the project development environment. It is essential that metrics collection and analysis not impede the primary development activities; thus, metrics collection and analysis should be as unobtrusive as possible. Periodically, the Process and Metrics team will publish reports about metrics collection and analysis tools available commercially for use on Contel projects.

Project managers may rely on this information or may request consultation with the Process and Metrics team about the special needs of the project at hand.

- **Help with cost and schedule estimation:** If requested, the Process and Metrics team will provide assistance with cost and schedule estimation. In concert with the data provided by project management, automated tools will be made available to suggest initial estimates of cost and schedule based on project characteristics. Where possible, estimates also will be made of the risk of missing cost and schedule goals. Tools can be used as development progresses to track revised estimates of cost and schedule. As custodian of a large set of such automated tools, the Process and Metrics team will suggest the most accurate tools based on the history of similar projects.
- **Collect appropriate level of metrics:** The project managers will oversee the collection of metrics.
- **Construct project database:** A database of project metrics will be designed, developed and populated, with assistance from the Process and Metrics team, if necessary. This database will be used for analysis where appropriate.
- **Evaluate cost and schedule:** When the project is complete, the initial and intermediate estimates of cost and schedule will be evaluated for accuracy. A determination will be made of the factors that may account for discrepancies between predicted and actual values.
- **Evaluate productivity and quality:** An overall assessment of project productivity and product quality will be made based on the metrics available.
- **Provide basis for future estimates:** Finally, the contents of the project database will be supplied to the Process and Metrics team for incorporation in a Contel corporate metrics database. This larger database will be used to provide historical information as a basis for estimation on future projects. In addition, it will be used to suggest the most appropriate tools and techniques for proposed projects.

4.2 BENEFITS OF USING THE METRICS SETS

The benefits of metrics collection and analysis are manifold. The increased understanding of the process and control of the project outweigh the effort needed to capture, store and analyze the information required. Objective evaluation of any new technique, tool or method is impossible without quantitative data describing its effect. Thus, the use of the recommended set of metrics should result in:

- Enhanced understanding of the process
- Increased control of the process
- A clear migration path to more mature process levels
- More accurate estimates of project cost and schedule
- More objective evaluations of changes in technique, tool or method
- More accurate estimates of the effects of changes on project cost and schedule

The understanding and experience gained by incorporating metrics are benefits that will carry over to future development projects. The implementation and use of metrics will help Contel produce a higher quality software product in a more efficient manner.

APPENDIX A

REFERENCES

1. W. Humphrey, *Managing the Software Process*, Addison Wesley, Reading, MA, 1989.
2. S. D. Conte, H. E. Dunsmore and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin-Cummings, Menlo Park, CA, 1986.
3. A. J. Albrecht, "Measuring Application Development Productivity", *Proceedings of the Joint SHARE/GUIDE Symposium*, October 1979, pp. 83-92.
4. H. P. Schultz, *Software Management Metrics*, Mitre Technical Report M88-1-ESD-TR-88-001, Bedford, MA, 1988.

CTC TECHNICAL DOCUMENT REPLY FORM

WE ARE INTERESTED IN YOUR COMMENTS ON THIS PUBLICATION. PLEASE
TAKE A MOMENT TO FILL OUT AND RETURN THIS FORM TO THE ADDRESS
BELOW. THANK YOU FOR YOUR INTEREST!

DATE _____ DOCUMENT NO. CTC-TR-89-017

DOCUMENT TITLE: RECOMMENDATIONS FOR AN INITIAL SET OF
SOFTWARE METRICS.

AUTHOR(S): SHARILAWRENCE PFLEEGER

YOUR NAME _____

TITLE _____

ADDRESS/LOCATION _____

CITY/STATE/ZIP _____

TELEPHONE NUMBER (____) _____

1. HOW DID YOU RECEIVE THIS DOCUMENT?

CTC MAILING LIST _____

SPECIAL REQUEST _____

CTCONTACT ORDER FORM _____

2. DID YOU FIND IT HELPFUL/USEFUL FOR YOUR NEEDS?

Y____ N____ SOMEWHAT____

3. SUGGESTIONS / COMMENTS _____

Please remove this form from the publication and mail it to Attn: Diane Weldin,
Publications Manager, Contel Technology Center, 15000 Conference Center Dr., PO Box
10814, Chantilly, VA 22021-3808.